Lösungen 35. BwInf

Betriebssystem : Ubuntu 16.04/Linux

Programmiersprache : Java

Luftballons :

Lösungsansatz : Brute-Force, Kombinatorik

Da es nur 10 Speicherfächer gibt, kann man hier alles aufzählen und sich nachher das Beste heraussuchen. Es gibt 10! = 3.628.800 Möglichkeiten, und die sind mit Java schnell gefunden und verglichen. Dies ist so weil man erstmals sich für eines von 10 Speicherfächern entscheiden muss, das man leert. Dann gibt es noch neun, dann acht usw. und so ergibt sich 10!.

Sourcecode mit Kommentaren :

LAMA :

Lösungsansatz : Implementierung

Anfangs sucht sich das Programm zufällig LEDs aus, dann kommt man in ein Setup und kann alles selbst nach konfigurieren, LEDs ab und anwählen durch Klicken, und mit dem Drücken der Taste f wird das Setup abgeschlossen. Dann kommt man ins Spiel. Durch Klicken passiert je nach Modus das Jeweilige. Durch einen Rechtsklick kann man nun die Größe ändern und mit der Leertaste den Modus. Ändert man etwas, kommt man erneut ins Setup. Sind im Spiel alle LEDs aus, hat man gewonnen, was dann in grüner Schrift angezeigt wird.
Möchte man nochmal kann man auf r drücken, dann beginnt es von neuem.

Umsetzung :

Zuerst importiere ich die nötigen Bibliotheken. Dann kommt die Hauptklasse, in der die Funktion main ist, die zuerst ausgeführt wird. Diese lasse ich dann ein neues Objekt der Hauptklasse erzeugen. Nun wird im Deskriptor der Hauptklasse ein Fenster mit Tasten und Maus abfangen erzeugt und die beiden Bilder werden geladen. Sind sie nicht vorhanden, werden automatisch neue erzeugt. Auch wird eine Zeichenfläche erzeugt und gesetzt. Diese hat eine Methode paintComponent. Für jedes Frame(es gibt ca. 25 pro Sekunde) wird diese von einer Endlosschleife aufgerufen. In dieser Methode lasse ich, je nachdem ob man gerade im Setup, Spiel oder Gameover ist, oben den jeweiligen Text und je nachdem das LED-Array anzeigen lassen. Parallel dafür laufen die Funktionen, die Maus-Tasten-Klicks und Tasten-Klicks abfangen. Bemerken sie so einen, passiert je nachdem ob man gerade im Setup, Spiel oder Gameover ist, dass das LED-Array geändert wird, alles von vorne losgeht oder der Setup abgeschlossen wird.

Sourcecode mit Kommentaren :

```
//Bibliotheken importieren für Fenster usw.
import javax.swing.JFrame;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;
import javax.swing.JPanel;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Random;
import java.util.Arrays;
import java.awt.EventQueue;
import javax.swing.JFrame;

public class Lightsout extends JFrame implements MouseListener, KeyListener {
     protected BufferedImage ledon; //Bild LED an
     protected BufferedImage ledoff; //Bild LED aus
     protected boolean gameover=false; //Spiel vorbei ?
     protected boolean normal=true; //Klassischer Modus(umliegende LEDs) ?
     protected boolean setup=false; //Konfiguration ?
     Random zufall = new Random(); //Zufall
     protected int score=0; //Züge
     protected int sizex=5; //Größe X
     protected int sizey=5; //Größe Y
     protected int arraylength=sizex*sizey; //Länge des Arrays mit LEDs
     protected int state=0; //Status der LED, der später immer angepasst wird
     protected int dec=0; //Zahl, die später eine Zufallszahl werden wird
     protected Integer[] newelement; //Array "Neues Element", ein Array, das an erster Stelle x-
Koordinate der LED, an Zweiter y-Koordinate der LED und an Dritter Status der LED beinhalten wird
     Integer[][] leds = new Integer[arraylength][]; //Array "LEDs" initialisieren mit "arraylength"
     public void reload_lights(Integer[][] leds) { //"LEDs" Array füllen mit LEDs entsprechend Größe
X, Größe Y mit zufällig bestimmten Staten, die sich aber bei der Konfiguration noch ändern lassen
         for(int xpos = 0; xpos < sizex; xpos = xpos+1){ //X-Koordinaten entsprechend X-Größe
entlanggehen
              for(int ypos = 0; ypos < sizey; ypos = ypos+1){ //Für jede dieser X-Koordinaten noch
mal Y-Koordinaten entsprechend Y-Größe entlanggehen
                  zufall = new Random(); //Zufall
                  dec=zufall.nextInt(5); //Zufallszahl
                  if (dec==2) { //Wenn die Zufallszahl zwei ist, soll der Status der LED an sein
                      state=1; //Statusvariable auf an setzen
                  }
                  else { //Ansonsten soll er aus sein
                      state=0; //Statusvariable auf aus setzen
                  }
                  Integer[] newelement = {xpos,ypos+1,state}; //LED-Array mit soeben bestimmter X-
Koordinate, Y-Koordinate + 1, weil darüber ja noch Text kommen soll, und der Statusvariable
                  leds[(ypos*sizex)+xpos]=newelement; //Entsprechende Stelle des LED-Arrays mit
"Neuem Element" füllen
                  newelement=null; //"Neues Element" freigeben für Ersetzen
              }
          }
          setup=true; //Danach Konfiguration des generierten Levels
     }
     public void mousePressed(MouseEvent m) {
     }
     public void mouseClickedinSetup(MouseEvent m) { //Wenn Maus geklickt
         int mouseposx=(int) (m.getX()); //Maus X-Koordinate
         int mouseposy=(int) (m.getY()); //Maus Y-Koordinate
         if (m.getButton() == m.BUTTON1) { //Wenn "Standard" erste Maus-Taste geklickt
             for (int iter2=0 ; iter2 < leds.length ; iter2=iter2+1) { //Für LED in LEDs
                 if (mouseposx >= (leds[iter2][0]*20) && mouseposx < (leds[iter2][0]*20)+20) {
//Wenn LED x auf Maus x
                     if (mouseposy >= (leds[iter2][1]*20) && mouseposy < (leds[iter2][1]*20)+20)
{ //Wenn LED y auf Maus y
                         score=score+1;
                         if (leds[iter2][2]==0) { //Status der LED ändern
                             leds[iter2][2]=1;
                         }
                         else { //Status der LED ändern
                             leds[iter2][2]=0;
                         }
                     }
```

```
                }
            }
        }
    }
    public void mouseClickedinGame(MouseEvent m) { //Wenn im Spiel geklickt wird
        int mouseposx=(int) (m.getX()); //Maus X-Koordinate
        int mouseposy=(int) (m.getY()); //Maus Y-Koordinate
        if (m.getButton() == m.BUTTON1) { //Wenn "Standard" erste Maus-Taste geklickt
            for (int iter2=0 ; iter2 < leds.length ; iter2=iter2+1) { //Für LED in LEDs
                if (mouseposx >= (leds[iter2][0]*20) && mouseposx < (leds[iter2][0]*20)+20) {
//Wenn LED x auf Maus x
                    if (mouseposy >= (leds[iter2][1]*20) && mouseposy < (leds[iter2][1]*20)+20)
{ //Wenn LED y auf Maus y
                        score=score+1; //Mit score sind Züge gemeint, ein Zug mehr
                        if (leds[iter2][2]==0) { //Status dieser LED ändern
                            leds[iter2][2]=1;
                        }
                        else { //Status dieser LED ändern
                            leds[iter2][2]=0;
                        }
                        for (int iter3=0 ; iter3 < leds.length ; iter3=iter3+1) { //Was ist mit den
anderen LEDs
                             if (!(iter3==iter2)) { //Alle nur nicht diese
                                 if (normal==true) { //Wenn normaler Modus
                                     if (leds[iter3][0]-1==leds[iter2][0] && leds[iter3]
[1]==leds[iter2][1]) { //LED links ändern
                                         if (leds[iter3][2]==0) {
                                             leds[iter3][2]=1;
                                         }
                                         else {
                                             leds[iter3][2]=0;
                                         }
                                     }
                                     if (leds[iter3][0]+1==leds[iter2][0] && leds[iter3]
[1]==leds[iter2][1]) { //LED rechts ändern
                                         if (leds[iter3][2]==0) {
                                             leds[iter3][2]=1;
                                         }
                                         else {
                                             leds[iter3][2]=0;
                                         }
                                     }
                                     if (leds[iter3][1]-1==leds[iter2][1] && leds[iter3]
[0]==leds[iter2][0]) { //LED oben ändern
                                         if (leds[iter3][2]==0) {
                                             leds[iter3][2]=1;
                                         }
                                         else {
                                             leds[iter3][2]=0;
                                         }
                                     }
                                     if (leds[iter3][1]+1==leds[iter2][1] && leds[iter3]
[0]==leds[iter2][0]) { //LED unten ändern
                                         if (leds[iter3][2]==0) {
                                             leds[iter3][2]=1;
                                         }
                                         else {
                                             leds[iter3][2]=0;
                                         }
                                     }
                                 }
                                 if (normal==false) { //Wenn nicht normaler Modus
                                     if (leds[iter3][0]<=leds[iter2][0] && leds[iter3]
[1]<=leds[iter2][1]) { //Wenn LED im Rechteck liegt, ändern
                                         if (leds[iter3][2]==0) {
                                             leds[iter3][2]=1;
                                         }
                                         else {
                                             leds[iter3][2]=0;
                                         }
                                     }
                                 }
                             }
                        }
                    }
                }
            }
        }
        if (m.getButton() == m.BUTTON3) { //Wenn rechte Maustaste gedrückt
            for (int xpos2 = 0; xpos2 < 20; xpos2 = xpos2+1) { //Für Quadrat x 1-20
                for (int ypos2 = 0; ypos2 < 20; ypos2 = ypos2+1) { //Für Quadrat y 1-20
                     if (mouseposx >= xpos2*20 && mouseposx < (xpos2*20)+20) { //Wenn in Quadrat x
                         if (mouseposy >= (ypos2*20)+20 && mouseposy < (ypos2*20)+40) { //Wenn in
Quadrat y
                             sizex=xpos2+1; //Größe ändern
                             sizey=ypos2+1; //Größe ändern
                             arraylength=sizex*sizey; //Länge des Arrays anpassen
                         }
                     }
                }
            }
        }
```

```
            leds = new Integer[arraylength][]; //LEDs resetten
            reload_lights(leds); //LEDs resetten
            score=0; //Züge resetten
        }
    }
    public void mouseClicked(MouseEvent m) { //Entscheiden, welche Funktion aufgerufen wird,
jenachdem ob Setup ist oder nicht
        if (setup==false) {
            mouseClickedinGame(m);
        }
        else {
            mouseClickedinSetup(m);
        }
    }
    public void mouseReleased(MouseEvent m) {
    }
    public void mouseEntered(MouseEvent m) {
    }
    public void mouseExited(MouseEvent m) {
    }
    public void keyTyped(KeyEvent e) {
    }
    public void keyReleased(KeyEvent e) { //Taste losgelassen
        if (setup==false && gameover==false) { //Wenn kein Setup und Spiel nicht vorbei, also im
Spiel
            if (e.getKeyCode() == KeyEvent.VK_SPACE) { //Wenn Leertaste gedrückt
                if (normal==true) {
                    normal=false;
                }
                else {
                    normal=true;
                }
                leds = new Integer[arraylength][]; //LEDs resetten
                reload_lights(leds); //LEDs resetten
                score=0; //Züge resetten
            }
        }
        if (setup==true) { //Wenn setup
            if (e.getKeyCode() == KeyEvent.VK_F) { //Wenn f gedrückt ("fertig")
                setup=false; //Setup beenden
                score=0; //Züge resetten
            }
        }
        if (gameover==true) { //Wenn Spiel vorbei
            if (e.getKeyCode() == KeyEvent.VK_R) { //Wenn r gedrückt ("nochmal")
                leds = new Integer[arraylength][]; //LEDs resetten
                reload_lights(leds); //LEDs resetten
                gameover=false; //Das Spiel ist nicht mehr vorbei
                score=0; //Züge resetten
            }
        }
    }
    public void keyPressed(KeyEvent e) {
    }
    public Lightsout(){
         super("Lightsout");
         reload_lights(leds); //LEDs resetten

         try {
         ledoff = ImageIO.read(new File ("grafiken/ledoff.png")); //Bild für LED aus laden
         } catch(IOException bug) { //Wenns nicht klappt, Ersatzbild erstellen (das schöner als das
Original ist)
         ledoff=new BufferedImage(20, 20, BufferedImage.TYPE_INT_RGB);
         Graphics2D ledoff_graphics=ledoff.createGraphics();
         ledoff_graphics.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
         ledoff_graphics.setColor(Color.WHITE);
         ledoff_graphics.fillRect(0,0,20,20);
         ledoff_graphics.setColor(Color.GRAY);
         ledoff_graphics.fillOval(1,1,18,18);
         ledoff_graphics.setColor(Color.BLACK);
         ledoff_graphics.drawOval(1,1,18,18);
         ledoff_graphics.setColor(Color.WHITE);
         ledoff_graphics.fillOval(4,4,6,6);
         System.out.println("Ein Bild wurde nicht gefunden. Deshalb wurde jetzt ein Ersatzbild
erstellt.");
         System.out.println(bug);
         }
         try {
         ledon = ImageIO.read(new File ("grafiken/ledon.png")); //Bild für LED an laden
         } catch(IOException bug) { //Wenns nicht klappt, Ersatzbild erstellen (das schöner als das
Original ist)
         ledon=new BufferedImage(20, 20, BufferedImage.TYPE_INT_RGB);
         Graphics2D ledon_graphics=ledon.createGraphics();
         ledon_graphics.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
         ledon_graphics.setColor(Color.WHITE);
         ledon_graphics.fillRect(0,0,20,20);
         ledon_graphics.setColor(Color.RED);
         ledon_graphics.fillOval(1,1,18,18);
         ledon_graphics.setColor(Color.BLACK);
```

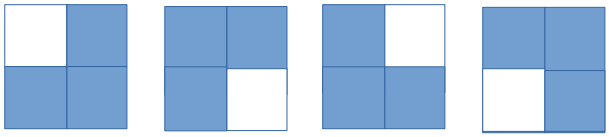
```
          ledon_graphics.drawOval(1,1,18,18);
          ledon_graphics.setColor(Color.WHITE);
          ledon_graphics.fillOval(4,4,6,6);
          System.out.println("Ein Bild wurde nicht gefunden. Deshalb wurde jetzt ein Ersatzbild
erstellt.");
          System.out.println(bug);
          }

          requestFocus(); //Fokus anfordern, also das das Fenster auf die Eingaben Zugriff hat
          addKeyListener(this); //Tasten abhören
          addMouseListener(this); //Maus abhören

          setContentPane(new Zeichenflaeche()); //Zeichenfläche setzen

          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Wenn es abgeschossen wird, schliesst
sich das Fenster

          setSize(400, 420); //Größe festlegen für 20x20 Spielfeld und 20px Schriftzug oben

          setResizable(false); //Größe des Fensters nicht änderbar

          setVisible(true); //Sichtbar machen

          while (true) { //Endlosschleife fürs Updaten des Fensters
              repaint();
              try {
              Thread.sleep(40); //Circa 25 FPS
              } catch(InterruptedException bug) {
              Thread.currentThread().interrupt();
              System.out.println(bug);
              }
          }
     }


     class Zeichenflaeche extends JPanel { //Zeichenfläche
        protected Font scorefont = new Font("Sans",Font.BOLD,11); //Standardschriftart
        protected Font gameoverfont = new Font("Sans",Font.BOLD,14); //Schrift, wenn man "gewonnen"
hat
        public Zeichenflaeche() {
        }
        public void setup(Graphics g) { //Was wird angezeigt wenn im Setup
           g.setFont(scorefont); //Standardfont
           g.setColor(Color.BLACK); //In Schwarz
           String ticker="Klick=ändern f=fertig"; //Infos
           for (int iter=0 ; iter < leds.length ; iter=iter+1) { //LEDs anzeigen
               if (leds[iter][2]==0) {
                   g.drawImage(ledoff, leds[iter][0]*20, leds[iter][1]*20, this);
               }
               if (leds[iter][2]==1) {
                   g.drawImage(ledon, leds[iter][0]*20, leds[iter][1]*20, this);
               }
           }
           g.drawString(ticker, 3, 15); //Infos anzeigen
        }
        public void inGame(Graphics g) { //Wenn im Spiel
           g.setFont(scorefont); //Standardfont
           g.setColor(Color.BLACK); //In Schwarz
           String scorestring="Züge: "+Integer.toString(score); //String für Züge
           String sizexstring="X: "+Integer.toString(sizex); //String für Größe X
           String sizeystring="Y: "+Integer.toString(sizey); //String für Größe Y
           String ticker=scorestring+" "+sizexstring+" "+sizeystring+" Rechtsklick=Größe
"+"Leertaste=Modus"; //Infos
           gameover=true;
           for (int iter4=0 ; iter4 < leds.length ; iter4=iter4+1) { //Wenn mindestens eine LED noch
an ist, ist das Spiel nicht vorbei. Ansonsten schon.
               if (leds[iter4][2]==1) {
                   gameover=false;
               }
           }
           for (int iter=0 ; iter < leds.length ; iter=iter+1) { //LEDs anzeigen
               if (leds[iter][2]==0) {
                   g.drawImage(ledoff, leds[iter][0]*20, leds[iter][1]*20, this);
               }
               if (leds[iter][2]==1) {
                   g.drawImage(ledon, leds[iter][0]*20, leds[iter][1]*20, this);
               }
           }
           g.drawString(ticker, 3, 15); //Infos anzeigen
        }
        public void gameOver(Graphics g) { //Wenn das Spiel vorbei ist
           g.setFont(gameoverfont); //Gewonnen-Schriftart
           g.setColor(Color.GREEN); //In Grün
           String wonstring="Du hast mit nur Züge/n gewonnen : "+Integer.toString(score)+"
r=nochmal"; //Infos
           g.drawString(wonstring, 3, 15); //Infos anzeigen
        }
        public void paintComponent(Graphics g) { //Jenachdem was gerade ist entsprechende Funktion
aufrufen, machts übersichtlicher
           if (setup==true) {
               setup(g);
```

```
            }
            else {
                if (gameover==false) {
                    inGame(g);
                }
                if (gameover==true) {
                    gameOver(g);
                }
            }
        }
    }
    public static void main(String args[]) {
           new Lightsout(); //Alles starten
    }
}
```

Rhinozelfant :

Lösungsansatz : Bilderkennung

Ich habe herausgefunden, das die Schuppen immer rechteckig zu sein scheinen und mindestens 2px hoch sind. Hier sind schematisch die vier Fälle für die vier Pixel einer 2x2 (der kleinstmöglichen) Schuppe dargestellt. Selbst wenn die Schuppen größer sind(bspw. 3x3 oder 4x2), so bestehen sie im Grunde doch aus 2x2 Schuppen. Blau bedeutet, dass die Farbe dieses Pixels mit dem zu prüfenden Pixel(Weiß) übereinstimmt. Weiß bedeutet, dass man in diesem Fall den Pixel einfärbt, da er als Teil einer Schuppe identifiziert wurde.
Also : Tritt mindestens einer dieser Fälle ein, wird der Pixel gefärbt.



Umsetzung :

Zuerst importiere ich die nötigen Bibliotheken. Dann kommt die Hauptklasse, in der die Funktion main ist, die zuerst ausgeführt wird. Diese lasse ich dann ein neues Objekt der Hauptklasse erzeugen. Nun wird im Deskriptor der Hauptklasse ein Fenster mit Tasten abfangen erzeugt das

Sourcecode mit Kommentaren :

```
import javax.swing.JFrame; //Einige Imports damit es das Fenster und die Keyprüfung usw. gibt
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.EventQueue;
```

```
import javax.swing.JFrame;
import java.util.*;

public class Rhinozelfant extends JFrame implements KeyListener {
     public BufferedImage rhinozelfant; //Rhinozelfantenbild
     public BufferedImage rhinozelfant_detected; //Eingefärbtes Bild
     public String path = "path"; //Pfad um das Bild zu laden
     public Scanner input = new Scanner(System.in); //Eingabe
     public int color; //Farbe des Pixels
     public int vergleichsfarbe; //Farbe der umliegenden Pixel
     public Integer width; //Breite des Bildes
     public Integer height; //Höhe des Bildes
     public int xscroll=0; //Scroll X
     public int yscroll=0; //Scroll Y

     public void keyTyped(KeyEvent e) { //Wird gebraucht von KeyListener, könnte man vermutlich
overriden
     }
     public void keyReleased(KeyEvent e) { //Wird gebraucht von KeyListener, könnte man vermutlich
overriden
     }
     public void keyPressed(KeyEvent e) { //Wird eine Taste gedrückt
         if (e.getKeyCode() == KeyEvent.VK_LEFT) { //Linke Pfeiltaste
             xscroll=xscroll-100; //100 Pixel nach links scrollen
         }
         if (e.getKeyCode() == KeyEvent.VK_RIGHT) { //Rechte Pfeiltaste
             xscroll=xscroll+100; //100 Pixel nach rechts scrollen
         }
         if (e.getKeyCode() == KeyEvent.VK_UP) { //Obere Pfeiltaste
             yscroll=yscroll-100; //100 Pixel nach oben scrollen
         }
         if (e.getKeyCode() == KeyEvent.VK_DOWN) { //Untere Pfeiltaste
             yscroll=yscroll+100; //100 Pixel nach unten scrollen
         }
     }
     public int get_at(int x,int y) { //Funktion, um nicht jedesmal rhinozelfant.getRGB(x, y)
schreiben zu müssen
         return rhinozelfant.getRGB(x, y);
     }
     public boolean check_at(int px, int py, int dir) { //Prüft einen der schematisch dargestellten
Fälle
         color=get_at(px,py);
         boolean retvar=false;
         if (dir==1) {
             if (color==get_at(px-1,py) && color==get_at(px,py-1) && color==get_at(px-1,py-1)) {
                 retvar=true;
             }
         }
         if (dir==2) {
             if (color==get_at(px+1,py) && color==get_at(px,py-1) && color==get_at(px+1,py-1)) {
                 retvar=true;
             }
         }
         if (dir==3) {
             if (color==get_at(px+1,py) && color==get_at(px,py+1) && color==get_at(px+1,py+1)) {
                 retvar=true;
             }
         }
         if (dir==4) {
             if (color==get_at(px-1,py) && color==get_at(px,py+1) && color==get_at(px-1,py+1)) {
                 retvar=true;
             }
         }
         return retvar;
     }
     public boolean check_schuppe(int px, int py) { //Prüft ob es eine Schuppe ist indem es die
Fälle prüft, allerdings die Fälle nicht prüft, wenn die zu prüfenden Pixel dort lägen, wo es keine
Pixel gibt
         boolean dir1=true;
         boolean dir2=true;
         boolean dir3=true;
         boolean dir4=true;
         boolean schuppe=false;
         if (px==0) {
             dir1=false;
             dir4=false;
         }
         if (py==0) {
             dir1=false;
             dir2=false;
         }
         if (px==width-1) {
             dir2=false;
             dir3=false;
         }
         if (py==height-1) {
             dir3=false;
             dir4=false;
         }
         if (dir1==true) {
             if (check_at(px,py,1)==true) {
```

```
                    schuppe=true;
                }
            }
            if (dir2==true) {
                if (check_at(px,py,2)==true) {
                    schuppe=true;
                }
            }
            if (dir3==true) {
                if (check_at(px,py,3)==true) {
                    schuppe=true;
                }
            }
            if (dir4==true) {
                if (check_at(px,py,4)==true) {
                    schuppe=true;
                }
            }
            return schuppe;
        }
        public Rhinozelfant() {
             super("Rhinozelfant");

             System.out.println("Pfad eingeben : ");
             path = input.nextLine(); //Pfad eingeben

             try {
             rhinozelfant = ImageIO.read(new File (path));
             } catch(IOException bug) { //Falscher Pfad
             System.out.println(bug);
             System.exit(9);
             }

             width=rhinozelfant.getWidth(); //Höhe ermitteln
             height=rhinozelfant.getHeight(); //Breite ermitteln
             if (width < 2 || height < 2) { //Ist das Bild kleiner als eine Schuppe
                 System.out.println("Was soll das ?");
                 System.exit(9);
             }
             rhinozelfant_detected=new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);

             for (int x = 0 ; x < width ; x=x+1) { //Für jede x-Koordinate des Bildes...
                   for (int y = 0 ; y < height ; y=y+1) { //Die y-Koordinaten entlanggehen
                       if (check_schuppe(x,y)==true) { //Prüfen, ob es eine Schuppe ist
                           rhinozelfant_detected.setRGB(x,y,Color.WHITE.getRGB()); //Weiß einfärben
                       }
                       else { //Wenn es keine Schuppe ist
                           rhinozelfant_detected.setRGB(x,y,get_at(x,y)); //Gleiche Farbe wie beim
Original nehmen
                       }
                   }
               }
             try { //Versuchen, das Ergebnis zu speichern
             File ergebnis = new File(path+"gescannt.jpg"); //Pfad+"gescannt.jpg"
             ImageIO.write(rhinozelfant_detected, "jpg", ergebnis); //Speichern als jpg
             } catch(IOException bug) {
             System.out.println(bug); //Wenns mal nicht klappt
             }

             requestFocus();
             setContentPane(new Zeichenflaeche());
             setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
             setSize(width,height); //Größe festlegen
             setResizable(false);
             setVisible(true);
             addKeyListener(this); //Tasten abhören

             while (true){
                 repaint(); //Erneut anzeigen
                 try {
                 Thread.sleep(250); //Viertel Sekunde warten
                 } catch(InterruptedException bug) {
                 Thread.currentThread().interrupt();
                 System.out.println(bug);
                 }
             }
        }
        class Zeichenflaeche extends JPanel{
            public Zeichenflaeche() {
            }
            public void paintComponent(Graphics g){
                g.drawImage(rhinozelfant_detected,xscroll,yscroll,this); //Geprüftes Bild anzeigen, mit
Scroll
            }
        }
        public static void main(String args[]){
               new Rhinozelfant(); //Ganzes starten
        }
}
```

Ergebnisse sind unter rhinozelfant/ergebnisse/ gespeichert.

Sprichwort :

Lösungsansatz : Osterberechnung mit Gauss, Unterschied berechnen

Mit der gausschen Formel wird Ostern berechnet, Weihnachten ist ganz gewöhnlich+Unterschied. Fallen sie zusammen : Jackpot.

Ausgabe : Im Jahre 11987 : Das orthodoxe Weihnachtsfest fällt mit dem katholischen Osterfest zusammen
Das katholische Weihnachtsfest und das orthodoxe Osterfest werden in den nächsten 99997985 Jahren nicht zusammenfallen.

Sourcecode mit Kommentaren :

```
public class Sprichwort {
     public int a; //Zwischenwerte
     public int b;
     public int c;
     public int k;
     public int M;
     public int d;
     public int N;
     public int p;
     public int q;
     public int e;
     public int O; //
     public int J=2016; //Jahreszahl
     public int gregOstern; //Greg Ostern
     public int juliOstern; //Juli Ostern
     public int weihnachten=-7;
     public int unterschied(int Jahr, int Monat) { //Weihnachten berechnen(Unterschied der Kalender)
            int JH=Jahr/100;
            if (Monat <= 2) {
                JH--;
            }
            int a=JH/4;
            int b=JH % 4;
            int TD=3*a+b-2;
            return TD;
     }
     public int gauss_computing(int Jahr, boolean julian) { //Ostern berechnen nach Gauss
            a = Jahr % 19;
            b = Jahr % 4;
            c = Jahr % 7;
            k = Jahr/100;
            if (julian==true) {
                M = 15;
                d = ((10 * a) + M) % 30;
                N = 6;
            }
            else {
                p = ((8 * k) + 13) / 25;
                q = k / 4;
                M = (15 + k - p - q) % 30;
                d = ((19 * a) + M) % 30;
                N = (4 + k - q) % 7;
            }
            e = ((2 * b) + (4 * c) + (6 * d) + N) % 7;
            O = (22 + d + e);
            if (O > 56) {
                O=56;
            }
            return O;
     }
     public Sprichwort() {
            int dif; //Unterschied
            int tbm; //Tage bis Ostern, da die gaussche Osterformel den xten März returnt
            while (true) {
                J=J+1; //Jahr hochzählen
                gregOstern=gauss_computing(J,false); //Ostern berechen
                dif=unterschied(J-1,12);
                if (J % 4 != 0 || J % 100 == 0) { //Schaltjahre !
                    tbm=31+28;
                }
```

```
                else {
                    tbm=31+29;
                }
                if (weihnachten+dif==tbm+gregOstern) { //Ist Weihnachten zum gleichen Datum ?
                    System.out.println("Im Jahre "+Integer.toString(J)+" : Das orthodoxe
Weihnachtsfest fällt mit dem katholischen Osterfest zusammen");
                    break;
                }
                if (J > 100000000) {
                    System.out.println("Das orthodoxe Weihnachtsfest und das katholische Osterfest
werden in den nächsten "+Integer.toString(J-2016)+" Jahren nicht zusammenfallen.");
                    break;
                }
            }
            J=2016;
            while (true) {
                J=J+1; //Hochzählen
                juliOstern=gauss_computing(J,true); //Ostern
                if (J % 4 != 0 || J % 100 == 0) { //Schaltjahre !
                    tbm=31+28;
                }
                else {
                    tbm=31+29;
                }
                if (weihnachten==tbm+juliOstern) { //Ist Weihnachten zum gleichen Datum ?
                    System.out.println("Im Jahre "+Integer.toString(J)+" : Das katholische
Weihnachtsfest fällt mit dem orthodoxen Osterfest zusammen");
                    break;
                }
                if (J > 100000000) {
                    System.out.println("Das katholische Weihnachtsfest und das orthodoxe Osterfest
werden in den nächsten "+Integer.toString(J-2016) +" Jahren nicht zusammenfallen.");
                    break;
                }
            }

    }


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

Radfahrspaß :

Lösungsansatz :

Wir lassen den Radfahrer durch-rollen und zählen dabei die Anzahl der flachen strecken als Möglichkeiten , zu bremsen oder zu beschleunigen.
Außerdem simulieren wir die Geschwindigkeit.

Wenn der Radfahrer irgendwann mehr oder gleich viele Möglichkeiten , zu bremsen oder zu beschleunigen als Geschwindigkeit hat, muss er auf einem teil der flachen Strecken mindestens beschleunigen , da er , wenn er auch auf diesen Strecken abbremsen würde 0 oder negative Geschwindigkeit haben würde.

Also suchen wir den teil der flachen Strecken, auf denen er nicht abbremsen darf , also beschleunigen muss

Damit er auf den übrig gebliebenen flachen strecken alles darf was er will , muss die Anzahl der übrig gebliebenen strecken kleiner als die Geschwindigkeit sein:

In einem Ungleichungssystem:

M sei die Anzahl der Möglichkeiten , zu bremsen oder zu beschleunigen

S sei die Geschwindigkeit = M – n

Z sei der teil der flachen strecken , auf denen er mindestens beschleunigen muss

$M – Z < M – n + Z \quad <=>$

$-Z < -n + Z \quad <=>$

$2Z > n \quad <=>$

$Z > n / 2$

da wir nur natürliche Lösungen suchen , und dem Radfahrer möglichst viele Möglichkeiten , zu bremsen oder zu beschleunigen lassen müssen , folgt:

$Z = \text{math.floor}(n / 2 + 0{,}5) \quad <=>$

$Z = \text{math.floor}((M - S) / 2 + 0{,}5)$

Denn M – S ist eine ganze Zahl , und eine ganze zahl durch zwei endet entweder auf ,0 oder auf ,5 , falls (M - S) / 2 auf ,0 endet , ist math.floor((M - S) / 2 + 0,5) um Eins größer als (M - S) / 2 , was heißt , dass die nächstkleinere ganze Zahl gleich groß ist wie (M - S) / 2 , falls (M - S) / 2 auf ,5 endet , ist math.floor((M - S) / 2 + 0,5) um Null Komma Fünf größer als (M - S) / 2 , was heißt , dass die nächstkleinere ganze Zahl kleiner ist als (M - S) / 2 .

Der Radfahrer ist Stehengebliebenen , wenn M < 0 ist , er also Möglichkeiten zum beschleunigen nutzen musste , die er nicht hatte

Wenn er Ankommt , nutzen wir folgendes Gleichungssystem um zu sehen , ob es möglich ist , die restlichen Möglichkeiten so zu verteilen , dass die Geschwindigkeit am Ende 0 ist:

M sei die Anzahl der Möglichkeiten , zu bremsen oder zu beschleunigen die der Radfahrer noch übrig hat.

S sei die Geschwindigkeit des Radfahrers am ende des Parcours

$| (M - |S|) / |(M - |S|)| = 1 |$
$| (M - |S|) \text{ modulo } 2 = 0 \quad |$

Denn (M - |S|) ist die Anzahl der überflüssigen Möglichkeiten , zu bremsen oder zu beschleunigen , falls die Anzahl der überflüssigen Möglichkeiten , zu bremsen oder zu beschleunigen kleiner als null ist , reicht die Anzahl der Möglichkeiten , zu bremsen oder zu beschleunigen nicht aus , um die Geschwindigkeit am Ende auf null zu bringen , und falls die Anzahl der überflüssigen Möglichkeiten , zu bremsen oder zu beschleunigen ungerade ist , muss man am Ende ein mal öfter abbremsen oder beschleunigen als nötig.

Die Anzahl der zum beschleunigen genutzten Möglichkeiten ist:

$S * -((S / |S|) / 2 - 0{,}5) + (M - |S|) / 2$

Da nur S mal Möglichkeiten zum beschleunigen genutzt werden , wenn S negativ ist und die Hälfte der überflüssigen Möglichkeiten ebenfalls zum beschleunigen genutzt wird.

Die Anzahl der zum beschleunigen genutzten Möglichkeiten ist:

S * ((S / |S|) / 2 + 0,5) + (M - |S|) / 2

Da nur S mal Möglichkeiten zum beschleunigen genutzt werden , wenn S positiv ist und die Hälfte der überflüssigen Möglichkeiten ebenfalls zum beschleunigen genutzt wird.

Nun wissen wir , wie oft der Radfahrer abbremst und wie oft er beschleunigt. Damit er nicht liegen bleibt , Wird er bei der Lösung zuerst so oft beschleunigen , wie er insgesamt beschleunigt , und dann wird er bei der Lösung so oft abbremsen , wie er insgesamt abbremst. Das wird natürlich nur gemacht , wenn er es überhaupt schaffen kann.

Sourcecode mit Kommentaren :

```
//Ntige Imports
import java.util.*;
import java.io.*;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;


public class Radfahrspass { //Hauptklasse
       public Scanner input = new Scanner(System.in); //Eingabe
       public String value; //Zwischenwert
       public int plusse=0; //Plusse
       public int minusse=0; //Minusse
       public int flatpos=0; //Möglichkeiten
       public int speed=0; //Geschwindigkeit
       public boolean possible=false; //Möglich ?
       public boolean liegengeblieben=false; //Liegengeblieben ?
       public BufferedReader parcours; //Leser
       public BufferedWriter ausgabe; //Ausgabe
       public String path; //Pfad
       public double y; //Zwischenwert
       public int z; //Zwischenwert
       public Radfahrspass() {
          System.out.println("Pfad eingeben : ");
          path = input.nextLine(); //Eingeben
          try {
          parcours = new BufferedReader(new InputStreamReader(new FileInputStream(path),"UTF-
8"),128); //Leser : Pfad,UTF-8 und jeweils 128 byte in Zwischenspeicher lesen
          z = parcours.read(); //Erstes Zeichen lesen
          while (z != -1) {
              value = new String(new char[] {(char)z}); //Zu String konvertieren
              if (value.equals("/")) { //Wenn bergauf
                  speed--; //Verlangsamen
              }
              if (value.equals("\\")) { //Wenn bergab
                  speed++; //Beschleunigen
              }
              if (value.equals("_")) { //Wenn flach
                  flatpos++; //Eine Möglichkeit zur Regulierung mehr
              }
              if (flatpos >= speed) { //Wenn es mehr Möglichkeiten als Geschwindigkeit gibt, dies
schließt aus das hier zu viele Minusse verwendet werden
                  y=Math.floor(((flatpos-speed)/2.0d)+0.5d); //Zusätzliche Plusse berechnen
                  plusse=plusse+(int)y; //Zu Plussen addieren
                  speed=speed+(int)y; //Geschwindigkeit erhöhen
                  flatpos=flatpos-(int)y; //Möglichkeiten abziehen
              }
              if (speed < 0) { //Wenn die Geschwindigkeit kleiner als null ist
```

```
                liegengeblieben=true; //Ist der Parcour unmöglich
                break; //Liegengeblieben.
            }
            if (speed==0) { //Wenn ich liegen bleibe
                if (flatpos > 0) { //Wenn ich davor hätte beschleunigen können
                    flatpos--; //Eine Möglichkeit weniger
                    plusse++; //Ein Plus mehr
                    speed++; //Und schneller.
                }
            }

            z = parcours.read(); //Zeichen lesen
            if (z==-1) { //Wenn zu Ende
                break; //Zu Ende
            }

        }
        if (speed==0 && liegengeblieben==false) { //Wenn er stehen, nicht liegen geblieben ist
            System.out.println("Stehengeblieben");
            if (flatpos % 2==0) { //Wenn man die Möglichkeiten gerecht aufteilen kann
                plusse=flatpos/2; //Aufteilen
                minusse=flatpos/2; //Aufteilen
                flatpos=0; //Keine Möglichkeiten mehr
                possible=true; //Möglich
            }
            else {
                possible=false; //Andernfalls unmöglich
            }
        }
        else if (liegengeblieben==false) { //Wenn man am Ende nicht 0 hatte, aber nicht liegen
geblieben ist
            if (flatpos > speed) { //Wenn man mehr Möglichkeiten hat, als man braucht
                if (speed < 0) { //Wenn man zu langsam ist
                    plusse=Math.abs(speed); //Alles nötige beschleunigen
                    flatpos=flatpos-speed; //Möglichkeiten
                    if (flatpos % 2==0) { //Wenn man den Rest aufteilen kann
                        plusse=plusse+(flatpos/2); //Aufteilen
                        minusse=minusse+(flatpos/2); //Aufteilen
                        flatpos=0; //Keine Möglichkeiten mehr
                        possible=true; //Möglich
                    }
                    else { //Ansonsten unmöglich
                        possible=false;
                    }
                }
                else { //Wenn man zu schnell ist
                    minusse=Math.abs(speed); //Verlangsamen, wo es geht
                    flatpos=flatpos-speed;
                    if (flatpos % 2==0) { //Wenn man den Rest aufteilen kann
                        plusse=plusse+(flatpos/2); //Aufteilen
                        minusse=minusse+(flatpos/2); //Aufteilen
                        flatpos=0; //Nix übrig
                        possible=true; //Möglich
                    }
                    else {
                        possible=false; //Ansonsten nicht
                    }
                }
            }
            if (flatpos==speed) { //Wenn man genauso viel zu schnell ist wie man verlangsamen
kann...
                possible=true;
                minusse=flatpos; //Einfach Minusse !
                flatpos=0;
                speed=0;
           }
        }
        parcours.close(); //Datei schliessen
        } catch(IOException bug) {
        System.out.println(bug);
        System.exit(9);
        }
        if (possible==true) { //Ausgabe...
            System.out.println("Dieser Parcours ist bewältigebar");
        }
        else {
            System.out.println("Dieser Parcours ist unmöglich");
        }
        if (possible==true)  {
            speed=0;
            try {
            parcours = new BufferedReader(new InputStreamReader(new FileInputStream(path), "UTF-
8"),128); //Einlesen
            } catch(IOException bug) {
            System.out.println(bug);
            System.exit(9);
            }
            try {
            ausgabe = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(path+"loesung.txt"), "UTF-8"),128); //Ausgabe
            z = parcours.read(); //Einlesen
```

```
            while (z != -1) {
                 value = new String(new char[] {(char)z}); //Konvertieren
                 if (value.equals("/")) { //Übernehmen
                     speed--;
                     ausgabe.write("/");
                 }
                 if (value.equals("\\")) { //Übernehmen
                     speed++;
                     ausgabe.write("\\");
                 }
                 if (value.equals("_")) { //Hier werden die Plusse und Minusse eingesezt.
                     if (plusse != 0) {
                         plusse--;
                         speed++;
                         ausgabe.write("+");
                     }
                     else {
                         if (minusse != 0) {
                             minusse--;
                             speed--;
                             ausgabe.write("-");
                         }
                     }
                 }
                 if (speed < 0) { //Wenn liegengeblieben
                     break;
                 }
                 z = parcours.read(); //Einlesen
                 if (z==-1) { //Abbrechen wenn zuende
                     break;
                 }
              }
              if (speed==0) { //Wenn am Ende stehen geblieben
                  System.out.println("Das ist nicht zu fassen ! Es hat geklappt !");

              }
              else {
                  System.out.println("Wieso klappt es nicht !");
              }
           parcours.close(); //Schließen
           ausgabe.close();
           } catch(IOException bug) {
               System.out.println(bug);
               System.exit(9);
               }
           }


       }
       public static void main(String args[]){
            new Radfahrspass(); //Deskriptor aufrufen
       }
}
```

Buhnenrennen :

Lösungsansatz :

Zuerst lässt man eine Brute-Force/Kombintorik den kürzesten Weg ermitteln. Diesen versucht Minnie zu approximieren, es sei denn, sie findet heraus, das sie es nicht kann, weil Max sie erwischen würde

Man lässt Minnie erstmal prüfen, welche Lücken sie erreichen kann, ohne das Max sie erwischt. Sie nimmt die Lücke, die mithilfe einer Bruteforce als Lücke die am kürzesten zum Ziel führt identifiziert wurde.
Max prüft selbst auch immer, ob er Minnie noch erwischen kann, wenn nicht, läuft er direkt auf die nächste Maxilücke zu.

Rotation :

Lösungsansatz : Brute-Force, Kombinatorik

Man zählt erstmal die zwei einfachsten Möglichkeiten auf : Drehung links/rechts
Wenn nun immer noch das Puzzle nicht gelöst ist, zählt man für jede dieser Möglichkeiten noch mal die Möglichkeiten auf : Drehung links/rechts.
Wenn ein Puzzle nicht lösbar ist , werden während dem Drehen der Box sich Zustände des inneren wiederholen , da man unendlich oft Weiterdrehen könnte , es aber nur endlich viele Zustände der Box gibt , und eine Periodizität eintritt wenn Zustände sich wiederholen.

Also drehen wir die Box nach Links und nach Rechts , prüfen , ob was raus-gefallen ist , und wiederholen die Prozedur , falls nichts raus-gefallen ist , falls sich bei einem Weg ein zustand wiederholt , wird dieser nicht weitergeführt.

Hier ist die besispielsweise Aufzählung für alle Kombis mit Länge 5. 0 sei links und 1 rechts.

00000
00001
00010
00011
00100
01000
01001
01010
01011
01100
01101
01110
01111
10000
10001
10010
10011
10100
10101
10110
10111
11000
11001
11010
11011
11100
11101
11110
11111