Lights Out

Kern | Schwartz | Zhanel

3AHIT 2016/17

**Anwendung**

Geübte Kompetenzen

Verwendung von APIs

Anwendungsprogrammierung: GUI- und Parallel Programmierung

Grundaufgabe

Die Implementierung eines Denksport-Spiels, in welchem es darum geht alle „Lichter“ auszuschalten. Sobald alle „Lichter“ deaktiviert sind, hat der Benutzer das Spiel gewonnen. Eingeschaltete Lichter werden mit der Farbe weiß, und ausgeschaltete mit der Farbe schwarz realisiert.

Team

Bilde Gruppen von 3-5 Personen. Jedes Mitglied dieses Teams muss seine eigenen Klassen implementieren, testen und den Code dokumentieren. Jedes Mitglied des Teams kann die Komplexität seiner Klasse nach Belieben anpassen.

**Benotung**

Ein Team kann mit einem (Großteiles) funktionsfähigem Programm an der Benotung teilhaben. Jedes Team wird nach den Programmteilen im Git-Hub benotet. Hier findet man auch die Dokumentation unseres Fortschritts.

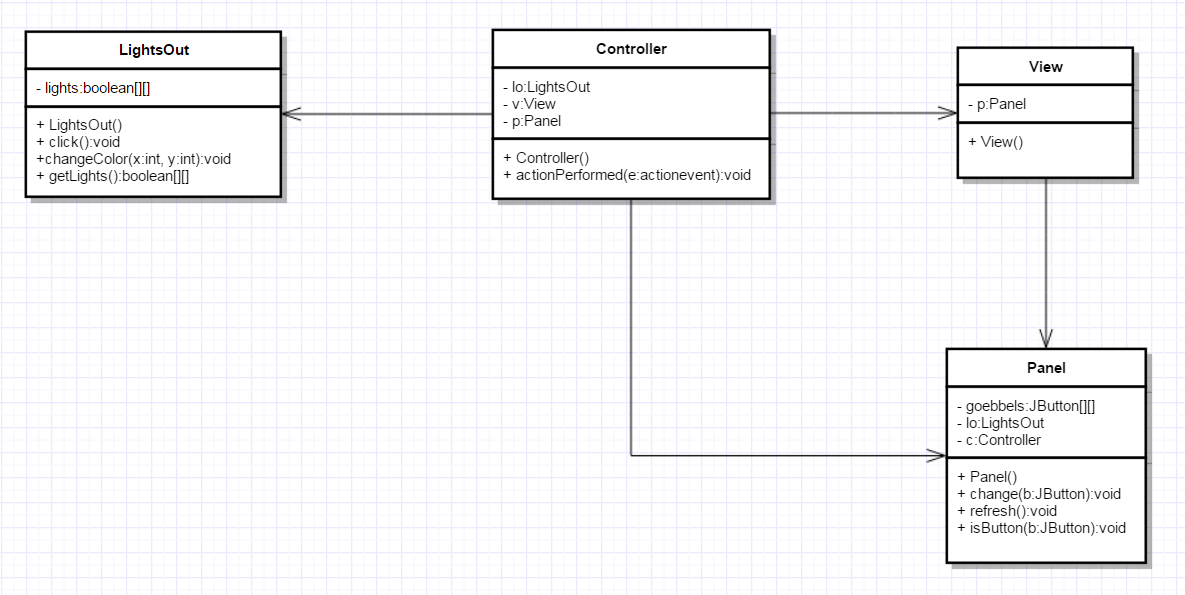
Abgabe

Jedes Team soll eine eigene Lösung zur Implementierung des Codes finden. Der Dokumentation muss ein UML-Klassendiagramm beigelegt werden. Die Lösung wird als Link zum GitHub abgegeben. Hier sieht man auch jenen Fortschritt der gemacht wurde und alle Veränderungen die getätigt wurden.

Nach der Abgabe des Links ist es immer noch möglich das Programm zu erweitern.

**Entwurf**

UML-Design



**Errungenschaften**

Panel

‘change’

Unser Panel enthält die „change“ Methode, welche die changeColor aus unserem Model (LightsOut) aufruft, um den Status der Buttons zu ändern.

**public** **void** change(JButton b){

**for**(**int** i=0; i<5; i++){

**for**(**int** j=0; j<5; j++){

**if**(**this**.goebbels[i][j] == b){

**this**.lo.changeColor(i, j);

}

}

}

}

‘refresh’

Die Methode “refresh” aktualisiert das Panel nach jedem gedrückten Button.

**public** **void** refresh(){

**boolean** licht[][] = lo.getLights();

**for**(**int** i=0; i<5; i++){

**for**(**int** j=0; j<5; j++){

**if**(licht[i][j])

goebbels[i][j].setBackground(Color.***white***);

**else**

goebbels[i][j].setBackground(**new** Color(78,78,78));

}

}

}

View

‘View’

In der View wird das Panel hinzugefügt. Die View dient hierbei als Fenster, in dem sich das Panel ausbreiten kann.

**public** View(Panel p, LightsOut lo, Controller c){

**this**.p = p;

**this**.c = c;

**this**.lo = lo;

**this**.setTitle("LightsOut");

**this**.setSize(1000, 1000);

**this**.setLocationRelativeTo(**null**);

**this**.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

**this**.add(**this**.p);

**this**.setVisible(**true**);

**this**.repaint();

}

LightsOut

‘click’

Unser Model (LightsOut) wird zum einen verwendet, um zufällige Buttons einzuschalten und auch auszuschalten.

**public** **void** click(){

**int** random = (**int**)(Math.*random*()\*24);

**for**(**int** i = 3; i <= random; i++){

**int** x = (**int**)(Math.*random*()\*4);

**int** y = (**int**)(Math.*random*()\*4);

changeColor(x, y);

}

}

‘changeColor’

Unsere „changeColor“ Methode wird verwendet, um den Status des Buttons zu ändern, wobei true = Licht an, false = Licht aus.

**public** **void** changeColor(**int** x, **int** y){

**if**(x != 0 && x != 4 && y != 0 && y != 4) {

**this**.lights[x][y] = !**this**.lights[x][y];

**this**.lights[x+1][y] = !**this**.lights[x+1][y];

**this**.lights[x-1][y] = !**this**.lights[x-1][y];

**this**.lights[x][y+1] = !**this**.lights[x][y+1];

**this**.lights[x][y-1] = !**this**.lights[x][y-1];

} **else** **if**(x == 0 && y != 0 && y != 4){

//…

}

}

Controller

‘actionPerformed’

Die „actionPerformed“ Methode überprüft welcher Button gedrückt wurde und weißt diesem eine Funktion zu.

**public** **void** actionPerformed(ActionEvent e) {

**if**(**this**.p.isStart((JButton)e.getSource())){

**this**.lo.click();

}

}

**Zeitaufwand**

|  |  |  |  |
| --- | --- | --- | --- |
| Aufgabe Person | Kern | Schwartz | Zhanel |
| Model | 0 | 10 min | 3h |
| View | 0 | 3 h | 0 |
| Controller | 0 | 3h | 2h |
| ReadME | 10min | 0 | 3min |
| Protokoll | 4h | 0 | 2h |
| Gesamt: | 4h 10min | 6h 40min | 7h 3min |