Lights Out

Zhanel | Schwarz | Kern

3AHIT 2016|17

Inhalt

Aufgabe..……………………………………………………………………………………………………………………………….2

Geübte Zuständigkeit ……………………………………….……………………………………………………….2

Anwendungen...…………………………………….………………………………………………………………….2

Team…………………………………………………………………………………………………………………………………….2

Benotung………………………………………………………………………………………………………………………………3

Abgabe………………………………………………………………………………………………………………………3

Kommunikation…………………………………………………………………………………………………………4

Zeitaufwand…………………………….……………………………………………………………………………………………5

UML-Design…..………………………………………………………………………………………………………………………6

Verrichtung…..…………………………………………………………………………………………………………….……7-11

Model..…………………………………………………………………………………………………………..………7/8

click………………………………………………………………………………………………………………..7

changeColor…………….……………………………………………………………………………………..8

Panel..…………………………………………………………………………………………………………..……………9

refresh……..…………….…………………….………………………………………………………………..9

color…………….…………………….…………………………………………………………………………..9

View...…………………………………………………………………………………………………………..…………10

View…………….…………………….…………………………………………………………………………10

Controller..…………………………………………………………………………………………………..………….11

actionListener…………….…………………………………………………………………………………11

Testen………………………………………………………………………………………………………………………………...11

Anwendung

Geübte Zuständigkeit

Verwendung von API´s

Anwendungs Programmierung: GUI- und Parallel Programmierung

Anwendungen

Dieses Spiel namens „Lights out“ soll dazu fähig sein auf einer graphischen GUI Oberfläche mit einem 5x5 Feld mit Buttons in Weiß und Schwarz alle Buttons Schwarz zu bekommen. Dies wird durch einen Mausklick auf den Button ermöglicht. Wenn man es schafft alle Buttons schwarz zu bekommen gewinnt man das Spiel.

Team

Bilde Gruppen von 3-4 Personen pro Gruppe. Jedes Team Mitglied muss seine eigene zuvor besprochene Klasse programmieren. Zum Beispiel einer macht den Controller, einer das Model und einer die View.

Benotung

Ein Team kann mit einem voll funktionstechnischem Programm zur Benotung antreten.

Jedes Team-Mitglied bekommt eine Individuelle Note welche auf die Dokumentation im Git-hub basiert. Dort ist jeder individuelle Programmteil durch den Namen sichtbar.

Abgabe

Jedes Team muss ein eigenes Programm welches sich von allen anderen Teams unterscheidet abgeben. Wenn dies nicht der Fall ist dann gibt es einen erheblichen Punkteverlust.

Weiters muss der Teamarbeit ein UML-Diagramm beigelegt werden.

Das Programm muss als .ZIP Datei abgegeben werden und noch dazu muss der Link von dem Git-hub vorhanden sein.

Man kann die Datei bis zum Abgabelimit hochladen.

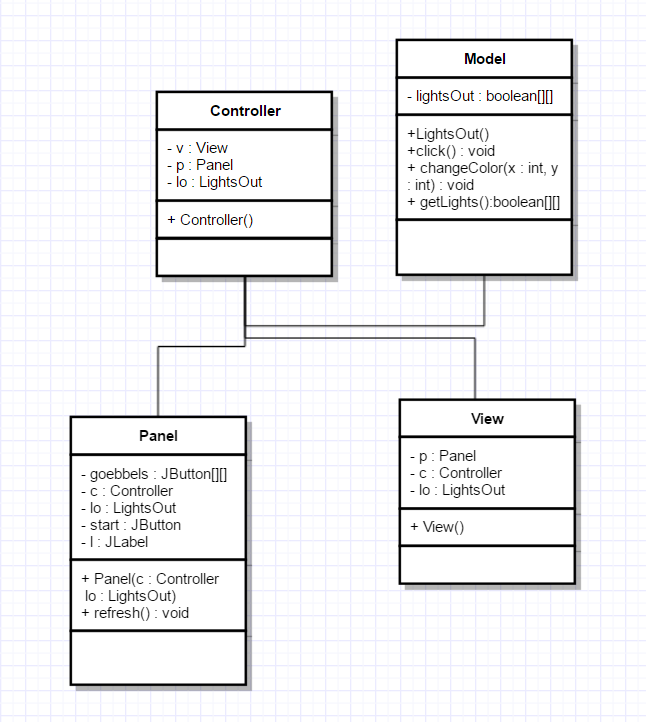
Kommunikation

Während des Projekts gab es immer einen Kommunikation zwischen den Teammitgliedern um das voranschreiten des Programm zu besprechen und sicher zu gehen das alle ihre Aufgabe verstehen und Lösen können.

Zeitaufwand

|  |  |  |  |
| --- | --- | --- | --- |
| Aufgabe | Zhanel | Schwarz | Kern |
| Model | 3h | 0 | 0 |
| View | 0 | 3h | 0 |
| Controller | 2h | 2h | 0 |
| ReadMe | 0 | 0 | 10 min |
| Protokollation | 0 | 0 | 4h |
| Gesamt: | 5h | 5h | 4.10h |

Overall: 14.10h

UML-Design

Verrichtung

Model

click

Wir verwenden unsere Methode Click um auf dem Knopfdruck die Farben unserer Buttons zu verändern.

**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);

Was wir von dieser Methode zurück bekommen sind 4 veränderte Farben. 3 rund um den gedrückten Button und 1 und zwar der Button auf den man drückt.

changeColor

In dieser Methode wird beschrieben wie bzw. wo genau sich die Farben ändern sollen.

**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){

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

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

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

**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];

…

Dies ist die Art des Algorithmus den wir verwenden um zu bestimmen welche genauen Buttons nun gefärbt werden sollen.

}

Panel

change

Diese Methode ändert die Farben der Buttons auf der GUI.

**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

Wie der Name bereits verrät wird diese Methode verwendet wenn die Farbe der Buttons verändert wird das dies auch sichtbar und aktualisiert wird.

**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])

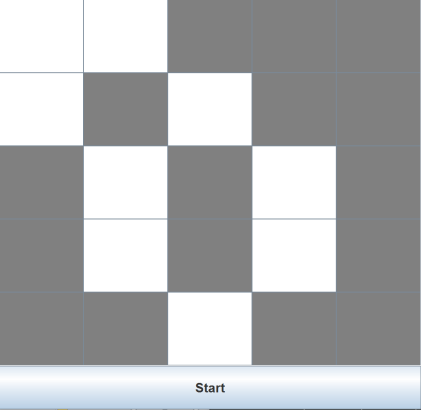
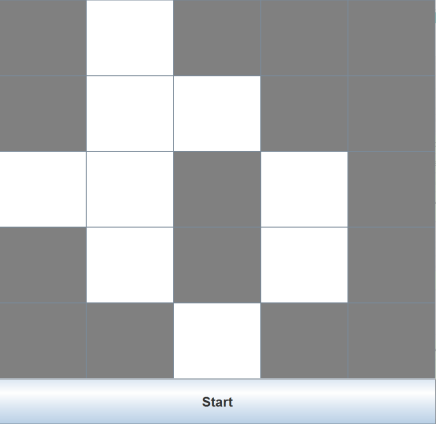
**this**.goebbels[i][j].setBackground(**Color**.***white***);

**else**

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

}

}

 }=>



View

View

Hier verwenden wir nur den Konstruktor um das Fenster (Frame) zu erstellen.

**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**);

}

Controller

actionPerformed

Hier verwenden wir ebenfalls nur den Konstruktor um die Buttons mittels dem ActionListener an zu sprechen und die Farben der Buttons zu ändern. Wir haben auch einen einzelnen Start Button um das Spiel zu beginnen.

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

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

}

**if**(**this**.p.isJButton1((**JButton**)e.getSource())){

**this**.p.change(**this**.p.getJButton1());

}

**if**(**this**.p.isJButton2((**JButton**)e.getSource())){

**this**.p.change(**this**.p.getJButton2());

}

**if**(**this**.p.isJButton3((**JButton**)e.getSource())){

**this**.p.change(**this**.p.getJButton3());

}

Testen

Um das Programm zu testen haben wir das Spiel mehrmals gestartet und gelöst um zu sehen was passiert wenn man was einen Button drückt oder wenn man alle Felder weiß bzw. schwarz hat. Weiters haben wir den Start Button öfter gedrückt um zu sehen ob immer ein neues Feld wie geplant generiert wird.