2016

# Dokumentation LightsOut

HASENBERGER|MAZZOLINI|REICHMANN|SEEMANN

3AHIT | 17.11.2016

# Inhalt

## Die Aufgabenstellung

Das Spiel „Lights Out“ soll im Team programmiert und getestet werden. Die ganze Aufgabe muss über Github verwaltet werden und dokumentiert werden.

## Spielprinzip Lights Out

Das Spielfeld besteht aus 25 Leuchtfeldern die in 5 Reihen und 5 Spalten aufgeteilt sind. Zu Beginn des Spiels leuchten manche Felder gelb und manche sind schwarz. Der Spieler kann durch klicken die Felder aus oder einschalten. Bei einem Klick werden die benachbarten Felder und das geklickte Feld invertiert. Das heißt leuchtende Felder werden schwarz und schwarze Felder werden eingeschalten. Ziel ist , wie der Name schon sagt, alle Felder auszuschalten. Am Ende müssen alle Felder schwarz sein. Aber Achtung: Die Zeit wie lange der Spieler gebraucht hat wird auch gestoppt.

## Team

Folgende Personen sind Teammitglieder:

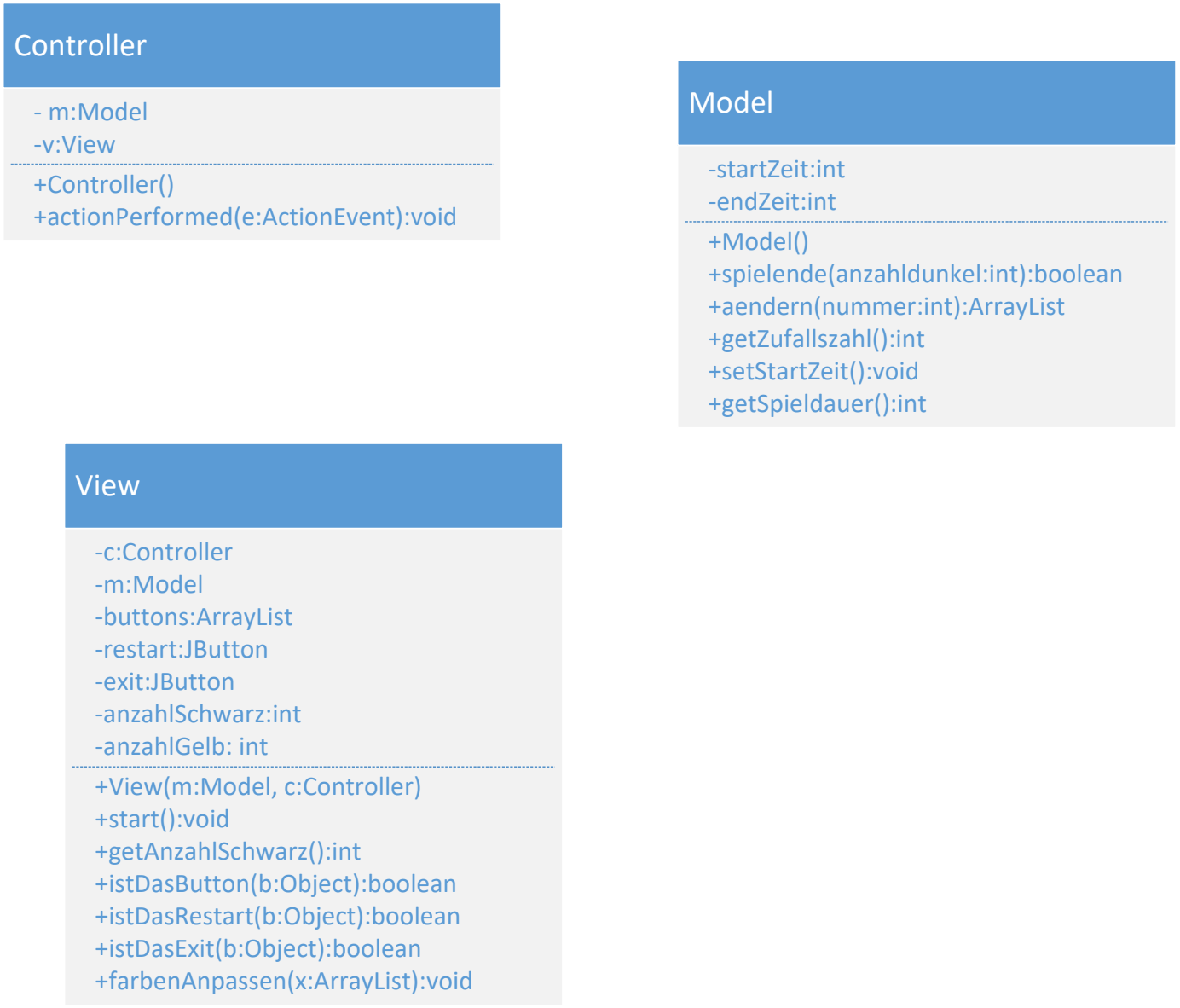
- Alexander Hasenberger
- Paul Mazzolini
- Adrian Reichmann
- Manuel Seemann

## Aufgabenverteilung

Die Aufgaben wurden folgendermaßen verteilt:

Hasenberger: Model, Dokumentation
Mazzolini: View, Controller , Model
Reichmann: Dokumentation , Controller , Model , View
Seemann: Github (Abgabe), Model, View

# UML


Controller
- m:Model
-v:View
+Controller()
+actionPerformed(e:ActionEvent):void
Model
-startZeit:int
-endZeit:int
+Model()
+spielende(anzahldunkel:int):boolean
+aendern(nummer:int):ArrayList
+getZufallszahl():int
+setStartZeit():void
+getSpieldauer():int
View
-c:Controller
-m:Model
-buttons:ArrayList
-restart:JButton
-exit:JButton
-anzahlSchwarz:int
-anzahlGelb: int
+View(m:Model, c:Controller)
+start():void
+getAnzahlSchwarz():int
+istDasButton(b:Object):boolean
+istDasRestart(b:Object):boolean
+istDasExit(b:Object):boolean
+farbenAnpassen(x:ArrayList):void

# Klassen

## GUI/View

Die GUI (Graphical User Interface) dient zur Ausgabe des Spiels.

Folgende Attribute wurden in der View verwendet.

```
/**
 * Attribute
 */
  private Controller c; //Der Controller
private Model m; //Das Model

private ArrayList<JButton> buttons; //ArrayList mit den Buttons
private JButton restart; //Button zum restarten des Spiels
private JButton exit; //Button zum schließen des Spiels

private int anzahlSchwarz; //Anzahl der schwarzen/licht aus Felder
private int anzahlGelb; //Anzahl der gelben/licht an Felder
```

Der Konstruktor sieht wie folgend aus:

```
/**
 * Konstruktor
 *
 * @param m das Model
 * @param c der Controller
 */
public View(Model m, Controller c){
	this.m = m;
	this.c = c;

	this.restart = new JButton("Restart");
	this.exit = new JButton("Exit");
	this.buttons = new ArrayList<JButton>();
	for (int i = 0; i < 25; i++) {
		this.buttons.add(new JButton(i+""));
		this.buttons.get(i).setFont(new Font("Arial", Font.PLAIN, 0));
	}

	JPanel b = new JPanel();
	b.setLayout(new GridLayout(0, 5));
	Iterator<JButton> iter = buttons.iterator();
	while(iter.hasNext()){
		b.add(iter.next());
	}

	JPanel s = new JPanel();
	s.add(this.restart);
	s.add(this.exit);

	this.setTitle("Light Out Game");
  this.setSize(500, 500);
  this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  this.setLayout(new BorderLayout());
```

```
        this.add(new JLabel("Light Out Game"), BorderLayout.NORTH);
        this.add(b, BorderLayout.CENTER);
        this.add(s, BorderLayout.SOUTH);

        //Adden zum ActionListener
        Iterator<JButton> i = buttons.iterator();
             while(i.hasNext()){
                   i.next().addActionListener(this.c);
             }
             restart.addActionListener(this.c);
             exit.addActionListener(this.c);

        this.setVisible(true);
      }
```

Um das Spiel jederzeit lösbar zu machen werden jedes Mal beim Start des Programms zuerst alle Buttons, auf schwarz gesetzt und dann eine zufällige Anzahl auf hell gesetzt. Dies wird mittels der Methode start() gelöst.

```
      /**
       * start Methode, weißt den Buttons Farben zu, Zufällige Anzahl an
gelben/leuchteten Buttons, die restlichen schwarz
       * Wird zu Begin des Programmes aufgerufen
       */
      public void start(){
            //Alle Buttons bekommen zunächst die Farbe schwarz
            for (int i = 0; i < 25; i++) {
                  this.buttons.get(i).setBackground(Color.black);
            }

            //Zufällige Buttons bekommen die Farbe gelb
            int za = this.m.getZufallszahl();//Zufallsanzahl der Elemente, die
gelb werden
            LinkedList gelbe = new LinkedList(); //Enthält die Gelben, zur
späteren Zuweisung der Anzahl
            for(int i=0; i < za; i++){
                  int zz = this.m.getZufallszahl(); //Zufallszahl welches Element
gelb wird
                  if(!gelbe.contains(zz)){
                        gelbe.add(zz);
                        this.buttons.get(zz).setBackground(Color.yellow);
                  }
            }
            this.anzahlGelb = gelbe.size();
            this.anzahlSchwarz = this.buttons.size()-this.anzahlGelb;
      }
```

## Ereignissteuerung/Controller

Der Controller dient, dazu bei z.B.: einem Klick auf einen Button eine Methode aus dem Model aufzurufen.
Folgende Attribute wurden im Controller verwendet.

```
/**
 * Attribute
 */
private Model m; //Das Model
private View v; //Die View
```

Der Konstruktor sieht wie folgend aus:

```
/**
 * Konstruktor
 */
public Controller(){
	this.m = new Model();
	this.v = new View(this.m, this);
	this.v.start();
}
```

Die Methode actionPerformed(ActionEvent e) ruft bei Klicks auf Buttons Methoden aus dem Model auf. Diese Methode reagiert bei Klick auf den Button „restart“ indem sie ein neues Fenster erstellt. Bei Klick auf „exit“ schließt diese Methode das Programm. Wenn einer der Leuchtbuttons gedrückt wird ändert diese Methode die Farbe des Buttons, aktualisiert den Counter mit der Anzahl der schwarzen Buttons, und gibt im Fall des Spielendes eine Nachricht aus und startet das Spiel dann von neu.

```
	/**
	* Verwaltet die Ereignisse bei z.B. Klicks auf Interaktionselemente in der View/Fenster (Ereignissteuerung)
	*
	* @param e Referenz auf das ActionEvent-Objekt
	*/
	@Override
	public void actionPerformed(ActionEvent e) {
		//Wenn restart Button gedrückt wurde
		if(this.v.istDasRestart(e.getSource()) == true){
			this.v.setVisible(false);
			this.v = new View(this.m, this);
			this.v.start();
		}

		//Wenn exit Button gedrückt wurde
		if(this.v.istDasExit(e.getSource()) == true){
			System.exit(0);
		}

		//Wenn einer der Buttons des Spiels gedrückt wurde
		if(this.v.istDasButton(e.getSource()) == true){
			JButton test = (JButton) e.getSource();
```

```
		this.v.farbenAnpassen(this.m.aendern(Integer.parseInt(test.getText())));
                    if(this.m.spielende(this.v.getAnzahlSchwarz())){
                          JOptionPane.showMessageDialog(null, "Spielende! Sie
haben gewonnen! " + "Sie brauchten: " +this.m.getSpieldauer() + "Sekunden.");
                          this.v.setVisible(false);
                          this.v = new View(this.m, this);
                          this.v.start();
                    }
              }
        }
```

## Model

Das Model bietet die Funktionalität welche das Programm benötigt. Die Methoden aus dem Model werden im Controller aufgerufen.

Folgende Attribute wurden im Controller verwendet.

```
        /**
         * Attribute
         */
        private int startZeit;
        private int endZeit;
```

Der Konstruktor sieht aus wie folgt:

```
        /**
         * Konstruktor
         */
        public Model(){
              this.startZeit = 0;
              this.endZeit = 0;
        }
```

Die Methode spielende(int anzahldunkel) überprüft, ob die Anzahl an dunklen Lichtern gleich der Anzahl aller vorhandener Lichter ist. Wenn dies der Fall ist, dann wird true zurückgegeben und das Spiel ist zu ende.

```
        /**
         * Diese Methode prüft, ob alle Lichter aus sind und das Spielende erreicht
ist.
         * @param anzdunkel ist die Anzahl an dunklen Lichtern
         * @return <code>true</code> wenn das Spielende erreicht wurde
         *              <code>false</code> andernfalls
         */
        public boolean spielende(int anzahldunkel){
               if(anzahldunkel == 25){
                     return true;
               } else {
                     return false;
               }
         }
```

Des Weiteren überprüft die Methode aendern(int nummer) welcher Button gedrückt wurde und gibt somit die Buttons zurück, welche ebenfalls farblich geändert werden müssen.

```
	/**
	 * Die Methode aendern überprüft das value des geklickten Feldes und return dann die zu ändernden Felder.
	 * @param nummer
	 * @return ArrayList <Integer> Die zu ändernen Farben
	 */

	public ArrayList <Integer> aendern (int nummer){

		ArrayList <Integer> rueckgabe = new ArrayList <Integer>();
		//Hauptfeld mitte
		if(nummer >=6 && nummer <=8 || nummer >=11 && nummer <=13 || nummer >=16 && nummer <=18){
			rueckgabe.add(nummer -5);
			rueckgabe.add(nummer +5);
			rueckgabe.add(nummer -1);
			rueckgabe.add(nummer +1);
			rueckgabe.add(nummer);
		}
		//Linker Rand
		if(nummer == 5|| nummer == 10 || nummer == 15){
			rueckgabe.add(nummer -5);
			rueckgabe.add(nummer +5);
			rueckgabe.add(nummer +1);
			rueckgabe.add(nummer);
		}
		//Oberer Rand
		if(nummer >= 1 && nummer <= 3){
			rueckgabe.add(nummer +5);
			rueckgabe.add(nummer -1);
			rueckgabe.add(nummer +1);
			rueckgabe.add(nummer);
		}
		//Rechter Rand
		if(nummer == 9|| nummer == 14 || nummer == 19){
			rueckgabe.add(nummer -5);
			rueckgabe.add(nummer +5);
			rueckgabe.add(nummer -1);
			rueckgabe.add(nummer);
		}
		//Unterer Rand
		if(nummer >= 21 && nummer <= 23){
			rueckgabe.add(nummer -5);
			rueckgabe.add(nummer -1);
			rueckgabe.add(nummer +1);
			rueckgabe.add(nummer);
		}
		//Ecke links oben
		if(nummer == 0){
			rueckgabe.add(nummer +5);
			rueckgabe.add(nummer +1);
			rueckgabe.add(nummer);
		}
		//Ecke rechts oben
		if(nummer == 4){
```

```
                rueckgabe.add(nummer +5);
                rueckgabe.add(nummer -1);
                rueckgabe.add(nummer);
        }
        //Ecke rechts unten
        if(nummer == 24){
                rueckgabe.add(nummer -5);
                rueckgabe.add(nummer -1);
                rueckgabe.add(nummer);
        }
        //Ecke links unten
        if(nummer == 20){
                rueckgabe.add(nummer -5);
                rueckgabe.add(nummer +1);
                rueckgabe.add(nummer);
        }
        return rueckgabe;
}
```

Die Methode setStartzeit() setzt die Startzeit des Zeitcounters.

```
/**
 * Setzt die Startzeit des Spiels / der Runde im Spiel
 *
 */
public void setStartZeit(){
        this.startZeit = (int)(System.currentTimeMillis());
}
```

Die Methode getSpieldauer() liefert die Spieldauer zurück.

```
/**
 * Berechnet die Endzeit und zieht davon die Startzeit ab
 *
 * @return Die Laufzeit in Sekunden umgerechnet
 */
public int getSpieldauer(){
        this.endZeit = (int)(System.currentTimeMillis());
        return (int)((this.endZeit-this.startZeit))/1000;
}
```

## GUI

Die GUI wurde möglichst simpel und effizient designed. Die einzelnen Leuchtfelder haben wir durch Buttons die ihren Background ändern realisiert. Am Ende des Spiels wird eine simple Meldung ausgegeben.

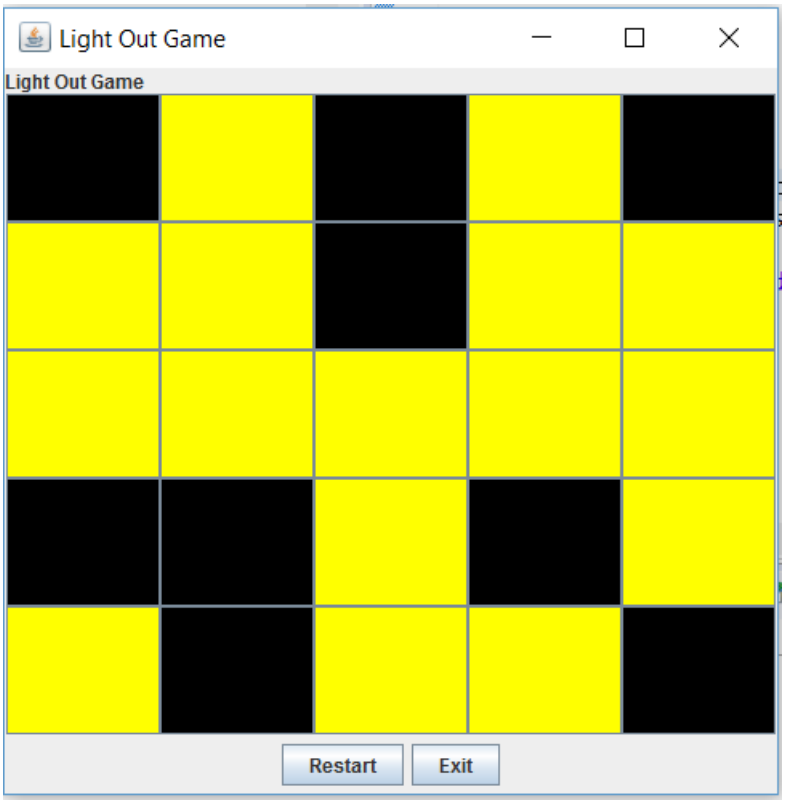



Abb.1.0:“Zu Start des Programm“

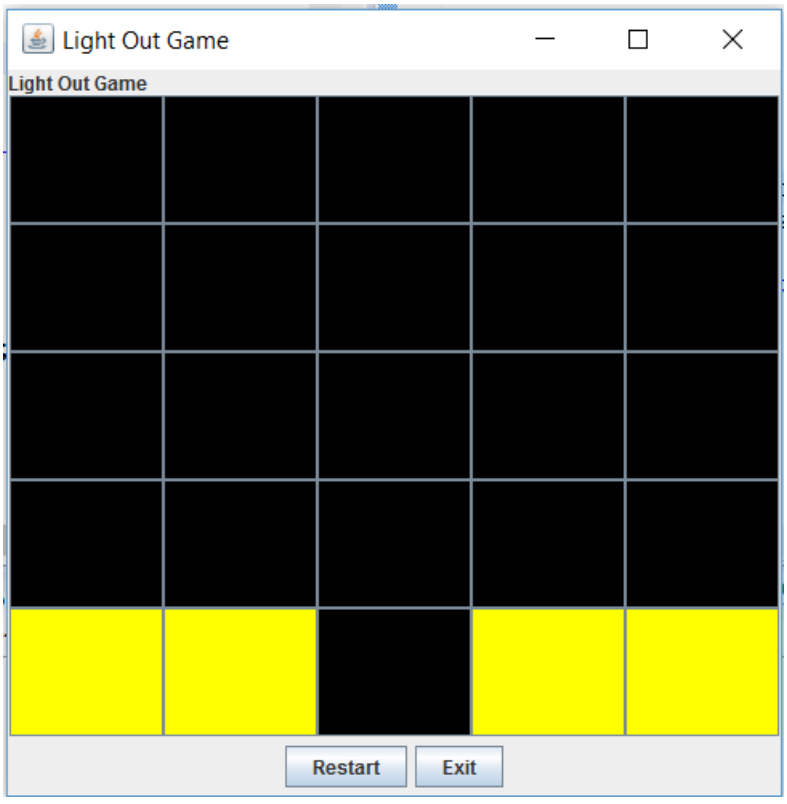



Abb.1.1:“Nach ein paar Klicks“

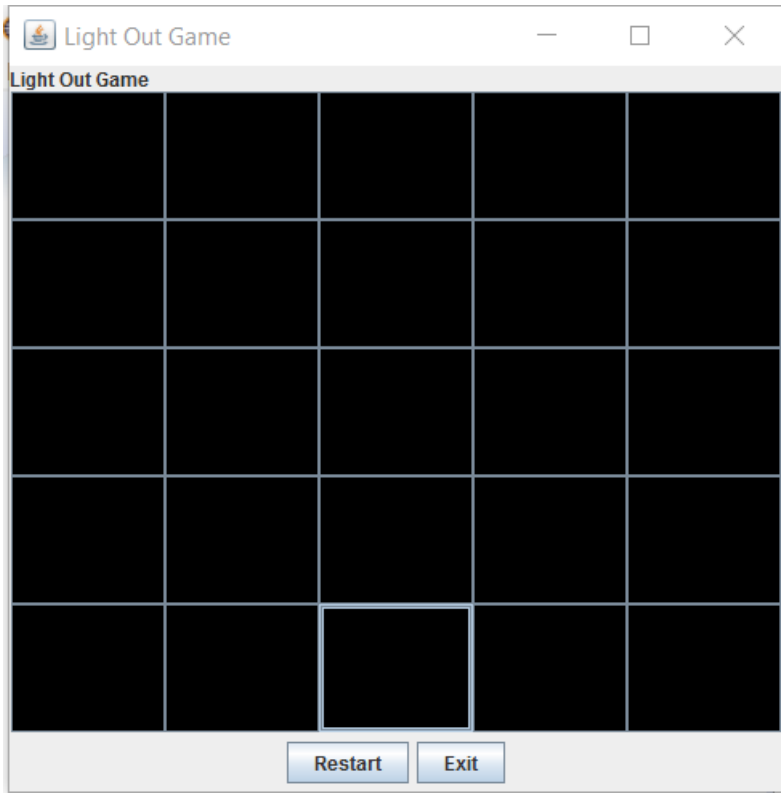



Abb.1.2:“So sieht das Feld zu Spielende aus“

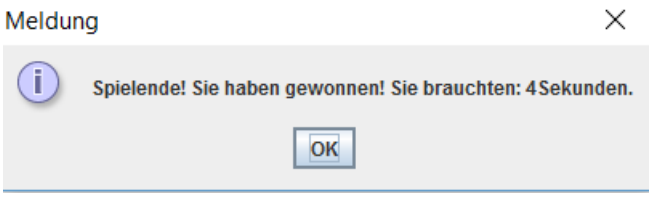



Abb.1.3:“Meldung wenn alle Lichter aus sind“