Lights Out

JaVA Team Aufgabe

Berger, May, Mühlehner

2014

Inhalt

[Aufgabe 2](#_Toc406572859)

[Trainierte Kompetenzen 2](#_Toc406572860)

[Grundlegende Aufgaben 2](#_Toc406572861)

[Erweiterte Aufgaben 3](#_Toc406572862)

[Team 3](#_Toc406572863)

[Aufwandsschätzung 3](#_Toc406572864)

[Konzept 4](#_Toc406572865)

[UML – Design 4](#_Toc406572866)

[Durchführung 4](#_Toc406572867)

[GUI 4](#_Toc406572868)

[Model 6](#_Toc406572869)

[Controller 6](#_Toc406572870)

[Installation 6](#_Toc406572871)

[Wie kann das Programm verwendet werden? 6](#_Toc406572872)

[Entwicklungsumgebung 6](#_Toc406572873)

[Tatsächlicher Aufwand 6](#_Toc406572874)

# Aufgabe

## Trainierte Kompetenzen

In dieser Aufgabenstellung wurden folgende Kompetenzen trainiert:

* GUI – Programmierung
* MVC – Architektur
* Arbeiten mit GitHub
* Teamarbeit

## Grundlegende Aufgaben

Die Aufgabenstellung ist, das Spiel "Lights out" zu programmieren.

Zuerst soll eine README Datei mit der Aufgabenstellung, dem Team und der Arbeitsaufteilung erstellt werden.

Danach soll das Programm umgesetzt werden. Dazu wird das Spiel nun ein wenig erklärt.

Das Spiel besteht aus einem 5x5 Gitter. Anfangs sind zufällig viele "Lichter" eingeschalten. Mit einem Klick auf ein Licht werden das betroffene Licht und alle umliegenden Lichter umgeschaltet. Diagonale Lichter sind davon nicht betroffen. Das Ziel ist es mit so wenig Klicks wie möglich alle Lichter auszuschalten.

Danach soll auch dieses Protokoll erstellt werden.

## Erweiterte Aufgaben

Je nach Interesse können erweiterte Aufgaben eingebaut werden. Dies ist nicht Pflicht und beeinflusst auch nicht die Note. Beispielsweise kann das Spiel für eine frei wählbare Größe, und nicht nur 5 x 5 programmiert werden.

## Team

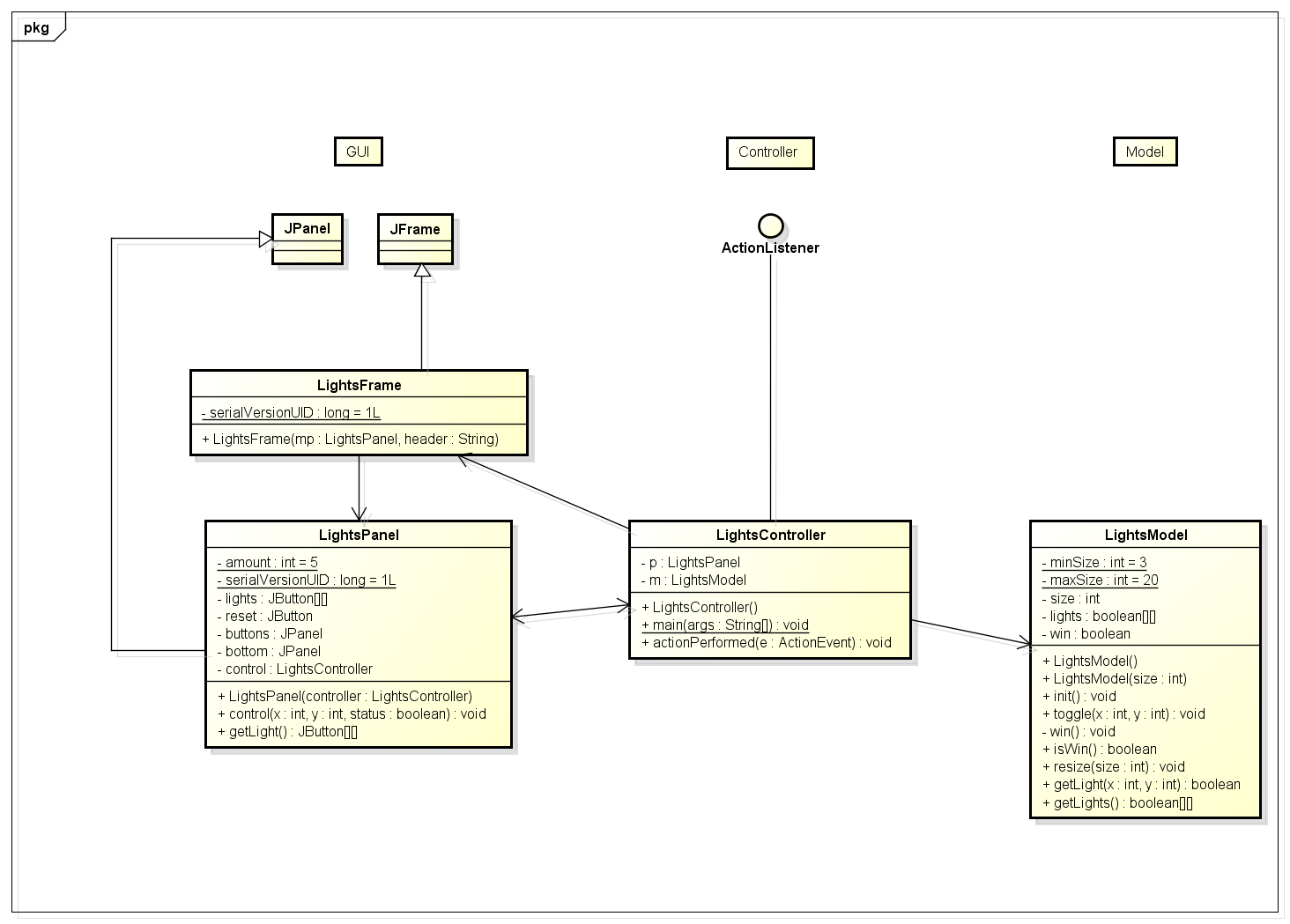
Das Team hat, nach Vorgabe, 3 Mitglieder. Die Aufteilung der Aufgaben ist grundsätzlich frei wählbar, jedoch sollte jeder einen gleichwertigen Teil des Programmes übernehmen. Da im Softwareentwicklungsunterricht derzeit mit dem MVC Prinzip gearbeitet wird, übernimmt jeder einen Teil davon. Jeder sollte sicherstellen, dass sein eigener Teil fehlerfrei ist (Unit – Testing), danach sollte das Zusammenspiel der Klassen geprüft werden. Schlussendlich auch das gesamte System an sich. Jedes Teammitglied kann zusätzlich erweiterte Aufgaben umsetzen, jedoch sind diese nicht Pflicht und fließen auch nicht in die Bewertung ein.

# Aufwandsschätzung

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | May | Mühlehner | Berger | Stunden |
| GUI | 0 | 0 | 3 | **3** |
| Erstellung | 1 | 0 | 0 | **1** |
| Controller | 0 | 3 | 0 | **3** |
| Model | 3 | 0 | 0 | **3** |
| Testen | 1 | 1 | 1 | **3** |
| Protokoll | 1 | 1 | 1 | **3** |
| h/Person | **6** | **5** | **5** |  |
|  |  |  | **Insgesamt:** | 16 |

# Konzept

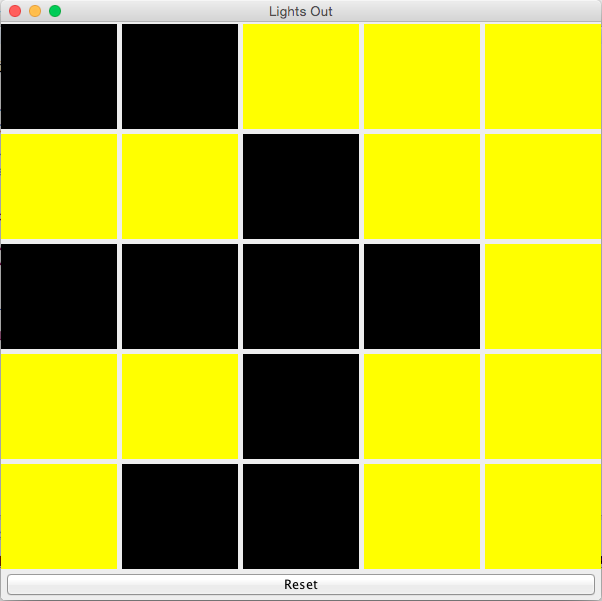
## UML – Design



# Durchführung

## GUI

Das User Interface wurde so einfach wie möglich designt um eine schnelle und einfach zu durchschauende Bedienung zu gewährleisten. Der einige Navigationsbutton in „Lights out“ stellt der Restart-Button am Boden des Programmes dar. Durch klicken kann ein neues Spiel gestartet werden, und alle bereits gespielte Züge werden auf einen zufälligen Stand eingestellt.



Da auf Mac OS X laufende Computer die Buttons nicht korrekt darstellten wurden, also keinen Hintergrundfarbe hatten, wurde eine Optimierung in diesem Sinne vorgenommen, welche auch Mac Benutzern eine einwandfreie Bedienung ermöglichen soll:

**for** (**int** x = 0; x < *amount*; x++) {

**for** (**int** y = 0; y < *amount*; y++) {

lights[x][y] = **new** JButton();

lights[x][y].setBackground(Color.***BLACK***);

**lights[x][y].setOpaque(true);**

**lights[x][y].setBorderPainted(false);**

lights[x][y].addActionListener(control);

lights[x][y].setActionCommand(y + " " + x);

buttons.add(lights[x][y], BorderLayout.***CENTER***);

}

}

## Model

Das Model wurde so designt, dass das Spielfeld größenveränderbar ist. Dazu wurden ein minSize und ein maxSize Attribut erstellt, welche die minimale und maximale wählbare Größe des Spiels regeln. Des Weiteren ist ein Attribut zur Speicherung der aktuellen Größe vorhanden. Alle Methoden sind ebenfalls auf eine variable Größe programmiert.

Zur Speicherung wurde ein zwei-dimensionales Array des Typen boolean gewählt, da die einzelnen Lichter einfach mit x und y Werten adressierbar sind.

In der toggle Methode wird das betroffene Licht umgeschaltet, hierbei werden Randsteine besonders behandelt.

In einigen Methoden kann eine IllegalArgumentException geworfen werden. Das passiert immer, wenn versucht wird ein Licht außerhalb des Spielfeldes zu adressieren.

Die init Methode wurde häufig verändert, da erkannt wurde, dass nicht alle Muster lösbar sind. Als Lösung zu diesem Problem wird das Spielfeld gelöst initialisiert und der Computer drückt wie ein Spieler zufällig 50-mal auf die Lichter. Sollte das Spielfeld dann zufällig schon gelöst sein, wird dieser Vorgang wiederholt. So wird garantiert, dass das Muster immer lösbar ist.

## Controller

# Installation

## Wie kann das Programm verwendet werden?

Zuerst muss auf dem Zielrechner Java, also eine JRE (Java Runtime Environment), installiert werden. Möglicherweise muss die JAVA\_HOME Variable in den Windows Umgebungsvariablen gesetzt werden.

Danach lässt sich die aktuellste Veröffentlichung über den nachfolgenden Link runterladen. Man lädt eine JAR Datei herunter, die dann am Rechner ausgeführt werden kann.

<https://github.com/dmay14/berger_may_muehlehner_lights_out/releases>

## Entwicklungsumgebung

Das Programm wurde auf Windows 7, Windows 8.1 und OS X Yosemite entwickelt. Die Entwicklungsumgebung war auf allen Betriebssystemen Eclipse. Die Versionen dieser Umgebungen waren Luna (4.4) und Juno (4.2).

# Tatsächlicher Aufwand