Minesweeper

Peter Martic/3AHITN/20-21

**Inhaltsverzeichnis**

[***Spielidee und Beschreibung*** 2](#_Toc65341286)

[***Klicken* auf leere Felder** 2](#_Toc65341287)

[**Highscore** 2](#_Toc65341288)

[***Technische Umsetzung(V1.0)*** 2](#_Toc65341289)

[**Logik** 2](#_Toc65341290)

[**GameButton** 2](#_Toc65341291)

[**Minesweeper** 2](#_Toc65341292)

[**MinesweeperGUI** 3](#_Toc65341293)

[**Klassendiagramm(V1.0)** 3](#_Toc65341294)

[***Technische Umsetzung(V1.1)*** 3](#_Toc65341295)

[**Logik** 3](#_Toc65341296)

[**Field** 3](#_Toc65341297)

[**GameButton** 3](#_Toc65341298)

[**Playground** 3](#_Toc65341299)

[**Klassendiagramm(V1.1)** 4](#_Toc65341300)

[4](#_Toc65341301)

[**Verwendete Algorithmen** 4](#_Toc65341302)

[**Index eines Feldes** 4](#_Toc65341303)

[**Bombenanzahl** 4](#_Toc65341304)

[***Technische Umsetzung(V1.2)*** 5](#_Toc65341305)

[**Logik** 5](#_Toc65341306)

[**Playground** 5](#_Toc65341307)

[**Klassendiagramm(V1.2)** 6](#_Toc65341308)

[***Technische Umsetzung(V1.3)*** 7](#_Toc65341309)

[**Logik** 7](#_Toc65341310)

[**GUI** 7](#_Toc65341311)

[**Playground** 7](#_Toc65341312)

[**Field** 7](#_Toc65341313)

[**Klassendiagramm(V1.3)** 8](#_Toc65341314)

# ***Spielidee und Beschreibung***

Bei Minesweeper geht es darum, dass alle Minen aufgedeckt werden ohne, dass man auf eine kommt. Wenn auf eine Mine gedrückt wird, dann ist das Spiel verloren und alle restlichen Minenfelder werden aufgedeckt. Wird hingegen ein freies Feld aufgedeckt, erscheint auf diesem die Anzahl der Minen in den angrenzenden Feldern.

**Kurz gefasst heißt dies:**

* Das Spiel wird beendet, wenn auf ein Minenfeld gedrückt wird, ansonsten kann weitergespielt werden
* Wird beim Aufdecken eines Feldes eine Zahl angezeigt, dann steht diese für die Anzahl der Minen, die in den benachbarten 8 Feldern verborgen sind. Anhand dieser Angabe kann abgeleitet werden, unter welchen der angrenzenden Feldern sich Minen befinden und in welchen Felder es keine gibt.

## ***Klicken* auf leere Felder**

* Es können Mienen im Feld markiert, falls man sich beispielsweise sicher ist, dass sich an bestimmten Feldern diese befinden. 🡪 Das kann mit einem Rechtklick ausgeführt werden.
* Falls auf eine leeres Feld gedrückt wird (ohne Minen oder Zahlen), dann passiert nichts.

### **Highscore**

Im Spiel gibt es auch einen Highscore mit der Bestzeit. Indem man seinen Namen vor dem Spiel eingibt, kann der Highscore mit der jeweiligen Zeit in eine Datei gespeichert werden.

# ***Technische Umsetzung(V1.0)***

## **Logik**

* In der Logik sind zurzeit die Klassen GameButton und MinesweeperGame vorhanden

### **GameButton**

Diese Klasse ist für die **Erstellung** eines Buttons zuständig, dies geschieht im Konstruktor der Klasse 🡪 Die Klasse wird in **MinesweeperGame** **verwendet**.

### **Minesweeper**

Sie ist zurzeit dafür verantwortlich, dass im **FlowLayoutPanel**, welches sich im GUI befindet, eine Fläche von 10x10 mit **Buttons** befüllt wird.

### **MinesweeperGUI**

Das ist die Benutzeroberfläche für den Spieler, wo die 10x10-Fläche erstellt wird. Zurzeit ist es nur möglich, wenn ein Button gedrückt wird, dass sich seine Hintergrundfarbe auf Rot ändert.

## **Klassendiagramm(V1.0)**

Klassendiagramm(V1.0)

# ***Technische Umsetzung(V1.1)***

## **Logik**

* In der Logik ist die Klasse **MinesweeperGame** entfernt worden, dafür aber wurden die Klassen **Playground** und **Field** hinzugefügt.

### **Field**

Diese Klasse wird in der Klasse Playground benötigt und stellt die **einzelnen Buttons**, die sich im **GUI** **befinden**, **logisch** da.

### **GameButton**

Diese Klasse wird benutzt, damit **im GUI** die **Buttons** dargestellt werden können.

### **Playground**

In dieser Klasse geschieht die ganze **Logik für das Spiel**, beispielsweise werden hier später die Methode erstellt, die für die Berechnung der Bombenanzahl nötig ist.

## **Klassendiagramm(V1.1)**



## 

Klassendiagramm(V1.1)

## **Verwendete Algorithmen**

### **Index eines Feldes**

Der **Index** eines Feldes kann mithilfe von **2 For-Schleifen**(einmal für die X- und Y-Koordinate) ermittelt werden, falls der Spieler auf einen Button drückt. Im nächsten Schritt wird eine if-Abfrage benötigt, wo untersucht wird, ob die **Koordinaten** des **gedrückten Buttons** mit den **aktuellen** der **For-Schleifen-Position übereinstimmen**, wenn es übereinstimmt, wird der jeweilige Index zurückgeliefert (int[]), ansonsten wird weiter gesucht.

### **Bombenanzahl**

Als erstes wird die Position des gedrückten Feldes benötigt(2 For-Schleifen, wie bei **„Index eines Feldes“**). Im 2. Schritt werden die umliegenden Felder des gedrückten Feldes nach Bomben untersucht(**max. 8 Bomben-Felder**) 🡪 würde z.B ein Feld, das sich in der Ecke oder irgendwo, wo nicht 8 umliegende Felder vorhanden sind, gedrückt, dann werden nur die Buttons **untersucht,** die **nicht über dem Spielfeld** sind . Im letzten Schritt werden diejenigen Felder(umliegend) **zusammengezählt**, die zu den **Bomben-Feldern** gehören🡪 es wird ein **int-Wert zurückgeliefert**.

# ***Technische Umsetzung(V1.2)***

## **Logik**

* In der Logik wurden die Methoden der Klasse Playground fertiggestellt **(Field und GUI bleiben größtenteils unverändert).**

### **Playground**

* Das Playground ist weiterhin der **Manager** für das GUI, aber die Klasse hat jetzt mehrere **Methoden** zur Verfügung.

#### **PlaceMines**

Diese Methode liefert **nichts** zurück. Hier wird dafür gesorgt, dass alle **10 Bomben für das 10x10-Feld platziert werden**. Dafür wird eine Do-While-Schleife gebraucht, damit sichergestellt werden kann, dass auch bei der 1. Platzierung wirklich eine Bombe platziert wird. In der Schleife wird mit der Random-Klasse werden 2 zufällige Koordinaten erstellt(für X- und Y-Koordinate). Im vorletzten Schritt wird mit der Methode **FieldHasMine()** geprüft, ob noch keine Bombe platziert wurde, falls dies stimmt wird für das Feld, welches zufällig durch die Random-Klasse ausgewählt wurde, die Eigenschaft „**HasBomb**“ auf true gesetzt und der int-Wert für die Anzahl der platzierten Bomben um eins erhöht. Zum Schluss wird geprüft, ob die Anzahl der platzierten Bomben die Zahl 10 überschritten hat, wenn das nicht der Fall ist, wird der Prozess wiederholt.

#### **CalculateSurroundingMines**

Diese Methode liefert **nix z**urück. Diese **Methode** ist dafür verantwortlich, dass die **Bomben**, **die sich umliegend um ein Feld befinden**, für das **aktuelle Feld** **errechnet werden**. Zuerst werden 2 For-Schleifen benötigt(X- und Y-Koordinate). Durch die For-Schleife wird es möglich gemacht, dass jedes Feld geprüft werden kann. Hier werden dann durch mehrere if-Abfragen geprüft, an welcher, der 8 möglichen Stellen, sich eine Bombe befinden könnte. Wird eine Bombe gefunden, wird die Eigenschaft „**SurroundingMines**“, für das aktuelle Feld, um ein eins erhöht.

#### **TagField**

Diese Methode liefert nichts zurück, hat **aber 2 Parameter für die X- und Y-Koordinate**. Diese Methode macht es möglich, dass ein **Feld** mit einem **Rechts-Klick - im GUI angezeigt – „getagged“** werden kann. Hier wird durch die übergebenen Koordinaten, jenes Feld „getagged“, welches diese Koordinaten beinhaltet, also die Eigenschaft „**IsTagged**“ des Feldes wird auf true gesetzt.

#### **FieldTagged**

Diese Methode liefert einen **Bool-Wert** zurück und hat dazu **2 Parameter die, die X- und Y-Koordinaten** beinhalten sollen. Hier wird geprüft, ob das Feld, dass durch die Koordinaten ausgewählt wird, „**getagged**“ ist oder nicht. Das einzige was hier **zurückgeliefert** wird, ist die **Eigenschaft** des Feldes „**IsTagged**“

#### **SurroundingMines**

Die Methode liefert einen **int-Wert** zurück und hat dazu **2 Parameter die, die X- und Y-Koordinaten** beinhalten sollen. Diese Methode soll es möglich machen, dass der Wert der umliegenden Bomben eines Feldes (**durch übergebene Koordinaten**), ausgegeben werden kann. Hier wird einfach die **Eigenschaft** „**SurroundingMines**“ des **Feldes** **zurückgeliefert**.

#### **FieldHasMine**

Die Methode liefert einen **Bool-Wert zurück und hat 2 Parameter die, die Koordinaten eines Feldes** beinhalten soll. In der Methode soll **geprüft** werden, ob das **aktuelle Feld eine Bombe beinhaltet**, dies kann möglich gemacht werden, indem nur die **Eigenschaft** „**HasBomb**“ des **Feldes** **zurückgeliefert** wird.

#### **Playground**

Das ist keine Methode, sondern der **Konstruktor der Klasse**. Hier werden nur **die 10 Felder initialisiert**, die **Bomben** **platziert** und die **umliegenden** **Minen** eines **Feldes** **berechnet**.

## Automatisch generierte Beschreibung**Klassendiagramm(V1.2)**

Klassendiagramm(V1.2)

# ***Technische Umsetzung(V1.3)***

## **Logik**

* In der Logik (Klasse „Playground“) wurde die Methode TurnOverField() hinzugefügt.

## **GUI**

* Hier wurde noch eine Methode TurnOverField dazugegeben, die aber noch leer ist.

#### **GameButtonClicked**

Diese Methode **liefert nichts** zurück. Die Methode ist **zuständig** dafür, wenn ein **Mausklick** auf ein **GameButton ausgeführt** wird, dass dann abgefragt wird, ob der Mausklick ein **rechter** oder **linker** war. Falls es ein **rechter** war, dann soll das **BackgroundImage des gedrückten Buttons** mit einer **Flagge „flag“ versehen** werden. Wenn es ein **linker** war, dann soll abgefragt werden, ob der **gedrückte GameButton** **eine Miene** hat oder **keine Bombe, aber umliegende Bomben**. Wenn eine **Miene vorhanden** ist, dann wird das **BackgroundImage des GameButtons** mit einer **Miene versehen**. Sind **umliegende Bomben vorhanden**, dann wird die **Anzahl errechnet** und mit dem **entsprechenden Bild** **versehen**.

## **Playground**

* Der Managerklasse wurde eine weitere Methode zugefügt.

#### **TurnOverField**

Diese Methode **liefert** einen **int-Wert zurück** und bekommt **2 int-Werte als Parameter** übergeben, die die **X- und Y- Koordinate eines Feldes** beinhalten. Weiters wird hier geprüft, ob ein Feld **umgedreht** wurde oder nicht(dies wird mit der **Eigenschaft „TurnedOver“** geprüft). Falls das Feld noch **nicht umgedreht** wurde, dann wird es **mithilfe dieser Methode umgedreht**, die Eigenschaft **„TurnedOver“** des Feldes auf **true** gesetzt und der **int-Wert 1 zurückgeliefert**. Ansonsten wird genau das Gegenteil gemacht, nur das statt am Ende eine 1 zurückgeliefert wird, eine **0 zurückgegeben wird**.

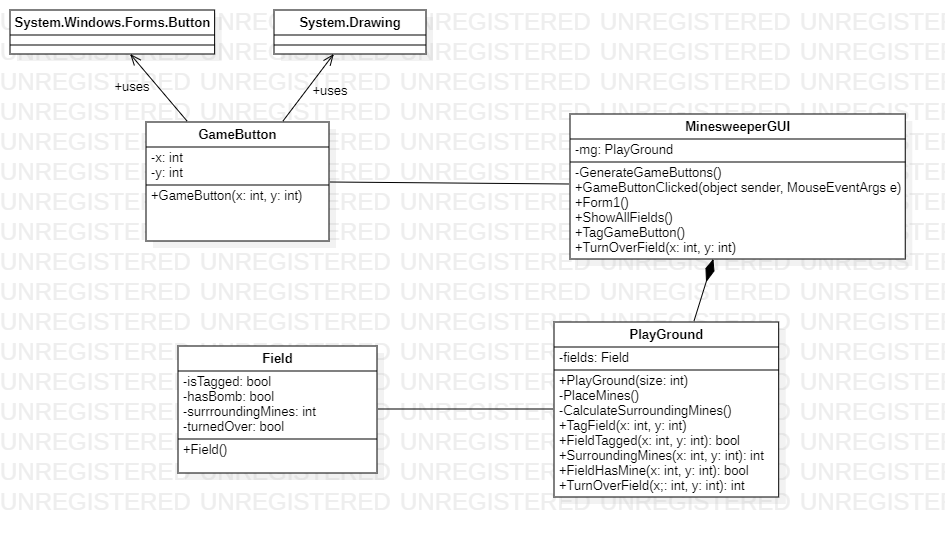
## **Field**

* Diese Klasse ist dazu zuständig, dass der optische Button vom GUI in der Logik mit der Klasse Playground arbeiten kann.

#### **TurnedOver**

Das ist keine Methode, sondern eine **Eigenschaft**, die mit dem Datentyp **bool** beschrieben ist. Diese Eigenschaft ist dafür verantwortlich, dass die Methode TurnOverField(), die sich im Playground befindet, prüfen kann, ob das **Feld umgedreht** wurde.

## **Klassendiagramm(V1.3)**



Klassendiagramm(V1.3)