Minesweeper Documentation

HHU-Programmierpraktikum SS2016 Projekt 6

Inhaltsverzeichnis

1 Graphical User Interface

Die Klassen, welche die grafische Benutzeroberfläche repräsentieren, befinden sich im Package `gui`.

1.1 `Main Window.java`

Diese Klasse repräsentiert das „Hauptfenster“, welches die zentrale GUI des Programmes darstellt. Sie enthält die `public static void main()`-Methode, welche beim Start des

Programmes aufgerufen wird und das Hauptfenster aufruft. Das Hauptfenster selbst enthält zugleich das Spielfeld als auch die Konfigurationsmöglichkeiten zum Spiel.

Diese bestehen aus einer `ComboBox<String> difficulty`, einer Auswahlbox, welche die vier Schwierigkeitsgrade „Easy“, „Intermediate“, „Hard“ und „Custom“ zur Auswahl anbietet. Daneben wird ein `TextField name` zur Eingabe des Spielernamens angeboten, welcher in den Highscore-Listen veröffentlicht wird. Sowohl die `ComboBox difficulty` und das `TextField name` werden von einem `Text`-Label beschrieben.

Zu guter Letzt existieren in diesem Menü noch ein `Button newGame`, welcher ein neues Spiel startet, und ein `Button highscores`, welcher das Highscore-Fenster aufruft.

1.1.1 Panels

Das Fenster selbst enthält ein großes `BorderPane`, in dessen `Top`-Feld ein `GridPane` zur Repräsentation der Einstellungsmenüs gesetzt wurde. Letzteres enthält wiederum zwei `GridPane`s, von denen das erste die regulären Einstellmöglichkeiten (Schwierigkeitsgrad, Name, Neues Spiel, Highscores anzeigen) bieten, und das zweite die erweiterten Einstellmöglichkeiten für den benutzerdefinierten Spielmodus repräsentiert.

Im `Center` des `BorderPane`s befindet sich das eigentliche Spielfeld, welche durch die Klasse `GamePane` realisiert wird. (*Siehe 1.2 GamePane.java*)

Im `Bottom`-Feld des `BorderPane` befindet sich wiederum ein weiteres `GridPane stats`, welches die „Statistiken“ des aktuellen Spiels repräsentiert. Dazu zählen die Anzahl der Minen, welche stets aktuell die Anzahl der im Spiel befindlichen Minen minus der bereits gesetzten Flags anzeigt, sowohl der Timer, welche laufend die Dauer des aktuellen Spiels anzeigt.

1.1.2 Benutzerdefinierte Spieleinstellungen

Der Nutzer kann durch Auswählen des Schwierigkeitsgrades „Custom“ ein benutzerdefiniertes Spiel erstellen. Dazu erscheint wird ein `GridPane difficultyMenu` sichtbar gemacht, welches drei `AutoCommitSpinner`-Objekte `xTilesSpinner`, `yTilesSpinner` und `minesSpinner` erzeugt. `AutoCommitSpinner` ist eine modifizierte Version des `Spinner`-GUI-Elements von JavaFX (*Siehe dazu 2.1 AutoCommitSpinner*).

In diesen drei Eingabefeldern kann der Nutzer die gewünschte Breite und Höhe des Feldes (in Anzahl Feldern) sowie die gewünschte Anzahl der Minen eingeben. Die Werte sind standardmäßig auf die Werte des letzten Spiels gesetzt, also z.B. auf die Einstellungen bei einem Spiel auf dem Schwierigkeitsgrad „Easy“, falls zuletzt ein solches gespielt / initialisiert wurde.

1.2 GamePane.java

Das eigentliche Spielfeld wird durch ein modifiziertes GridPane repräsentiert, welches jedes einzelne Feld im Spielfeld als ein Field-Objekt (*Siehe 1.3 Field*) speichert und im GridPane verwaltet. Es existieren Methoden zum Erzeugen eines neuen Spiels, zum Erzeugen eines neuen Spiels mit veränderten Einstellungen, zum Aufrufen des Spielstatusses „Gewonnen“ oder „Verloren“ sowohl zum Abfragen der aktuellen Spieleinstellungen.

1.3 Field.java

Diese Klasse repräsentiert ein einzelnes Feld im Spielfeld. Sie ist wiederum eine abgewandelte Form eines GridPanes, welche im Wesentlichen nur einen Button enthält. Dieser ist aktiv, wenn zugedeckt, und inaktiv, wenn aufgedeckt (Grafisch sichtbar).

Es wurde ein GridPane gewählt, statt nur einen Button zu verwenden, da sonst beim Deaktivieren des Buttons ebenfalls die enthaltenen Child-Elemente (z.B. eine Grafik, die die Bombe oder die Flagge repräsentiert, oder ein Text, welcher die Anzahl der Minen angibt) ebenfalls deaktiviert und somit ”transparentünd schlecht lesbar erscheinen. Da diese Elemente nun direkt auf das GridPane gesetzt werden können und nicht auf den Button ist diesem Problem vorgebeugt.

Das Feld erstellt beim Erzeugen automatisch einen Listener per Lambda-Expression, welcher beim Klick überprüft, ob die rechte oder die linke Maustaste gedrückt wurde. Im Falle der Linken Maustaste wird Field.open() aufgerufen, im anderen Fall Field.flag().

Das Feld speichert den Button button, welcher das grafische „Feld“ darstellt, booleans mine, flagged und hidden, welche respektive speichern, ob das Feld eine Mine, mit einer Flagge versehen und noch geschlossen / bereits geöffnet ist, sowie einen int neighbourMines, welcher speichert, wie viele Minen dieses Feld umgeben.

1.3.1 Field.open()

Die Methode Field.open() enthält die gesamte Logik, die beim Aufdecken des Feldes erforderlich ist. Es wird unterschieden, ob das Feld eine Bombe ist oder nicht. Wenn das Feld eine Bombe ist, wird abgefragt, ob dies der erste Klick auf ein Feld ist. Falls ja, wird die Bombe von dem Feld entfernt und es wird zufällig ein Feld gefunden, welches noch keine Bombe hat und auf dieses die Bombe gesetzt wird, anschließend wird das Feld aufgedeckt. Ist es nicht der erste Klick, so wird das Feld geöffnet, eine Bombe angezeigt, falls das Feld geflaggt war, die Flagge in die untere rechte Ecke verkleinert, und das Spiel als verloren markiert.

Ist das Feld keine Bombe, so wird das Feld selbst aufgedeckt. Ist die Zahl der angrenzenden Minen 0, so wird keine Zahl angezeigt und alle umliegenden Felder ebenfalls aufgedeckt. Ist die Zahl nicht null, so wird sie auf dem Feld angezeigt und keine weiteren Felder aufgedeckt. Der erste Klick ist in diesem Fall irrelevant.

1.4 HighscoreStage.java

Das Anzeigen der Highscores wird von der Klasse `HighscoreStage` übernommen. Wie der Name schon sagt, stellt diese Klasse eine spezifizierte Version der JavaFX-Klasse `Stage` dar, und initialisiert sich beim Erzeugen einer neuer Instanz dieser Klasse automatisch. Die Highscores werden bei Klick auf den Highscores-Button in einem eigenen Fenster angezeigt.

Das Fenster enthält ein großes `BorderPane root`, welches im `center` die eigentlichen Highscores, und im `Bottom` zwei Knöpfe zum Zurücksetzen der Highscores und zum Schließen des Fensters besitzt.

Die eigentlichen Highscores im `center` werden durch ein `GridPane` realisiert, welches einen `Text title` als Titel und ein `TabPane` bietet, um durch die Highscore-Tabellen der einzelnen Schwierigkeitsgrade zu schalten. Das `TabPane` enthält also vier `Tab`s, `easyTab`, `intermediateTab`, `hardTab` und `customTab`, welche jeweils eine Instanz `HighscoreTableView` `easyTable`, `intermediateTable`, `hardTable` bzw. `customTable` enthalten (*Siehe 1.5 HighscoreTableView.java*). Diese `HighscoreTableView`s sind die eigentliche grafische Darstellung der Highscores.

Die Highscores enthalten dabei die folgenden Daten, und sind somit recht ausführlich:

1. Highscore-Platzierung
2. Name
3. Startzeit
4. Spieldauer
5. Anzahl Spielzüge
6. Breite des Spielfelds (in Feldern)
7. Höhe des Spielfelds (in Feldern)
8. Anzahl der Minen

... wobei die letzten drei Werte nur für den benutzerdefinierten Schwierigkeitsgrad angezeigt werden. Der Benutzer ist in der Lage, die Datensätze nach den jeweiligen Feldern sowohl auf- als auch absteigend zu sortieren, sowie eine Reihenfolge festzulegen, um nach mehreren Feldern zugleich zu sortieren (z.B. vorrangig nach Spieldauer, bei gleicher Spieldauer nach Name, bei gleichem Namen nach Startzeit).

1.5 HighscoreTableView.java

Wie der Name schon sagt, handelt es sich hier erneut um eine Spezifikation einer JavaFX-Klasse, in diesem Falle der Klasse `TableView`. Bei der Erzeugung eines solchen Objektes werden automatisch die entsprechenden `TableColumn`s dem `TableView` hinzugefügt (die Spalten vorbereitet) und die Daten geladen (*siehe 4.2 Datenverwaltung*).

1.6 FacePane.java

Das FacePane stellt den Smiley dar, der in der oberen rechten Ecke des Hauptfensters angezeigt wird, und den Status des Spiels (laufend, verloren, gewonnen) repräsentiert. Es ist ein GridPane, welches die vier verschiedenen Smileys als ImageView enthält, und Methoden bereitstellt, um den sichtbaren dieser vier ImageViews zu wechseln.

2 Custom GUI

Für einige Funktionen meines Programms habe ich die von JavaFX bereitgestellten GUI-Objekte modifiziert. Diese Klassen befinden sich im Package metagui.

2.1 AutoCommitSpinner.java

Es gibt einen bekannten Bug in aktuellen Versionen von Java, welcher dafür sorgt, dass manuell per Tastatur eingegebene Werte in den Spinner-Elementen nicht übernommen werden. Das bedeutet, der Spinner trägt z.B. momentan den Wert 10. Nun wird per Tastatur der Wert 15 eingegeben, dieser wird aber aufgrund dieses Bugs nicht übernommen. Wird nun die Pfeil-nach-oben-Taste oder Pfeil-nach-unten-Taste des Spinners verwendet, um den Wert entsprechend zu erhöhen oder zu vermindern, wird weiterhin vom ursprünglichen Wert 10 ausgegangen, und somit z.B. die Werte 9 und 11 produziert, statt wie zu erwarten 14 oder 16. Genauso ist es beim Abfragen des Wertes des Spinners, dort würde weiterhin der Wert 10 ausgegeben werden, obwohl bereits 15 eingegeben wurde.

In dieser Modifikation sorgt ein TextListener dafür, dass bei einer Tastatureingabe im Spinner sofort der Wert des Textfeldes abgefragt wird. Es wird versucht, diesen Wert zu parsen, und als neuen Wert des Spinners zu übernehmen. Schlägt das Parsen fehl, weil z.B. ein nicht-numerischer Wert eingegeben wurde, so wird der Wert des Spinners automatisch auf den zuletzt akzeptierten Wert zurückgesetzt.

2.2 RowNumberCell.java

Diese Klasse repräsentiert eine modifizierte Version der TableCell, welche immer den Wert ihres Zeilenindexes enthält. Werden solche Zellen zu einem TableView hinzugefügt, so zeigen sie immer die Nummer der Zeile an, in der sie stehen. Auf diese Weise habe ich das Nummerieren der Highscore-Einträge realisiert.

3 Neues Spiel

- Beim Starten des Programmes wird automatisch ein neues Spiel mit den zuletzt gewählten Einstellungen (bei der letzten Verwendung des Programmes) erstellt. Ist das Programm zuvor noch nicht verwendet worden, bzw. sind keine lesbaren Speicherdaten vorhanden, wird automatisch ein neues Spiel auf Schwierigkeitsgrad „Easy“ erstellt.
- Ein neues Spiel wird automatisch beim Wechseln des Schwierigkeitsgrades auf „Easy“, „Intermediate“ oder „Hard“ erstellt,
- Beim Wechseln des Schwierigkeitsgrades auf „Custom“ wird kein neues Spiel erstellt, da hierzu zunächst die Einstellungen des benutzerdefinierten Spiels erforderlich sind. Der Nutzer muss mit einem Klick auf den „New Game“-Knopf ein neues Spiel starten.
- Das Spiel wird allgemein nur initialisiert, gestartet wird es erst, sobald der Nutzer das erste Feld anklickt. Erst dann wird der Timer gestartet.

3.1 Highscores

4 Interna

Diese Klassen realisieren interne Funktionen des Programmes und befinden sich im Wesentlichen im Package `meta`.

4.1 Timer

Der Timer wird durch die Klasse `TimerText` realisiert, welche ein JavaFX-`Text`-Objekt darstellt. Der Timer enthält also sowohl den Thread, der alle x Millisekunden die angezeigte Zeit aktualisiert, als auch das Textfeld, worin die Zeit angezeigt wird. Somit kann der Timer einfach als GUI-Element verwendet werden.

Der Timer speichert beim starten oder resetten eine Startzeit als `LocalDateTime`-Objekt. Es kann die Dauer mittels der Methode `TimerText.getDuration()` abgefragt werden, welche ein `java.time.Duration`-Objekt zurückgibt. Dazu wird ein neues `LocalDateTime`-Objekt mit der aktuellen Systemzeit erstellt, und die Differenz zwischen dieser Zeit und der Startzeit mittels der Methode `Duration.between()` berechnet und zurückgegeben. Die Startzeit kann ebenfalls durch einen getter Abgefragt werden, diese und die Duration werden ebenfalls für die Highscores verwendet.

4.2 Datenverwaltung

Die Daten werden in den Klassen `DataManager` und `Data` bereitgestellt. Dabei ist `DataManager` eine statische Klasse, welche jeweils das zur Laufzeit aktuelle `Data`-Objekt enthält. Dieses Objekt kann neu erzeugt, aus einer Datei geladen, in eine Datei geschrieben, oder im Programm abgefragt werden. Somit können sich die Klassen automatisch Daten holen, die sie brauchen, oder Daten schreiben.

Die Einstellmöglichkeiten im Hauptfenster speichern etwaige Änderungen z.B. immer sofort im `Data`-Objekt ab, wird dann ein neues Spiel erstellt, werden diese Daten automatisch aus dem `Data`-Objekt gelesen. Ebenfalls wird im `Data`-Objekt verfolgt, wie viele Flaggen gesetzt sind, wodurch der Minenzähler aktualisiert werden kann, ob das Spiel bereits gewonnen oder verloren ist, oder ob der erste Klick bereits gemacht wurde. Auch wird die Zahl der aufgedeckten Felder gezählt, dadurch kann festgestellt werden, ob das Spiel gewonnen wurde (alle Felder sind aufgedeckt). Diese Informationen werden an verschiedenen Stellen des Programmes gebraucht.

4.3 Aktualisierung der Spielinfos

Die Aktualisierung der Spielinfos wird im Wesentlichen von den `Field`-Feldern des Spielfeldes übernommen. Bei einem Klick und dem damit einhergehenden Aufdecken eines Feldes wird z.B. der Zähler der aufgedeckten Felder in dem `Data`-Objekt verändert, um zu verfolgen, ob das Spiel gewonnen wurde oder nicht. Die Zustände Gewonnen und Verloren werden ebenfalls von der `Field`-Klasse aus gesteuert. Ebenfalls wird beim Setzen eines Flags der entsprechende Counter im `Data`-Objekt und damit der Zähler am unteren Bildschirmrand aktualisiert.

5 Speichern + Laden

Um die Highscores und Einstellungen zu sichern, bietet die Klasse `DataManager` Methoden zum Speichern des `Data`-Objektes in eine Datei und zum Lesen aus einer Datei. Dazu wird ein `ObjectOutputStream` bzw. `ObjectInputStream` auf einen `FileOutputStream` bzw. `FileInputStream` gesetzt, welcher das Objekt serialisiert und in eine Datei schreibt bzw. aus ihr liest. Nach dem Einlesen eines gespeicherten `Data`-Objektes ist somit der alte Zustand des `Data`-Objektes mit allen Feldern wiederhergestellt, also sowohl den Einstellungen als auch den Highscores.

Da in der Datenstruktur der Highscores, `HighscoreEntry`, Instanzen der `javafx.beans.property.SimpleIntegerProperty`, `javafx.beans.property.SimpleStringProperty` und `javafx.beans.property.SimpleObjectProperty` verwendet werden, um die Werte zu speichern und das automatische Einlesen dieser Werte in die Highscore-Tabelle zu ermöglichen, entsteht ein Problem, denn diese Klassen sind nicht serialisierbar. Somit wäre es unmöglich,

das Data-Objekt mit den Highscores zu serialisieren.

Um dies zu umgehen, habe ich die Methoden Object HighscoreEntry.writeReplace() und Object HighscoreEntrySerializable.readResolve() implementiert, sowie die Klasse HighscoreEntrySerializable erstellt, welche der Klasse HighscoreEntry ähnelt, aber keine Schnittstellen aufweist und keine javafx.beans.property.SimpleIntegerProperty, javafx.beans.property.SimpleStringProperty oder javafx.beans.property.SimpleObjectProperty verwendet, sondern einfach nur Integer, Strings oder die entsprechenden Objekte direkt speichert. Somit ist diese Version der HighscoreEntry serialisierbar. Durch das Bereitstellen dieser Methoden werden diese automatisch beim Schreiben bzw. Lesen eines Objektes ausgeführt.

Die Methode Object HighscoreEntry.writeReplace() wird beim Schreiben des Data-Objektes aufgerufen und wandelt alle HighscoreEntry-Objekte, die im Data-Objekt gespeichert sind und geschrieben werden sollen, automatisch in HighscoreEntrySerializable-Objekte um. Die Methode Object HighscoreEntrySerializable.readResolve() wird beim Lesen eines Data-Objektes aus der Datei aufgerufen, und wandelt alle gespeicherten HighscoreEntrySerializable-Instanzen wieder zurück in HighscoreEntry-Objekte.