Dokumentation BomberHulk

Gruppe 23

0 Inhaltsverzeichnis

1 Klassen

Das Programm BomberHulk wurde in der Programmiersprache Java verfasst, da sich diese Sprache durch Systemunabhängigkeit und einfache Strukturierung mit Hilfe der Objektorientierung auszeichnet.

1.1 Menue

implementiert KeyListener

Das Programm wird durch die main-Methode in dem *Menue* initialisiert. Hier werden die entsprechenden Objekte zur Grundsteuerung erzeugt, sowie das GUI aufgebaut und verwaltet. Als Bibliotheken für diese werden sowohl die *swing*- als auch die *awt*-Bibliothek verwendet. Beides sind Standartbibliotheken des Java Development Kits 6.x (im Folgenden JDK genannt).
Zudem implementiert die Klasse das *KeyListener*-Interface, da die Tastatureingaben direkt im *Menue* interpretiert und, je nach Aktion, entweder sofort weiterverarbeitet oder an die *Map* weiter gegeben werden, da sich dort die Kollisionsabfragen befinden.
Die Zugriffe auf die Klassenvariablen sowie die entsprechenden Objekte erfolgt über Getter-/Setter-

Methoden, welche sich im unteren Teil des Quellcodes befinden. Sie sind durch Kommentare entsprechend gekennzeichnet (was auch für alle folgenden Getter-/Setter-Methoden gilt). Ebenso werden die Soundeffekte hier verwaltet (siehe *Sound*).
Bei der Netzwerknutzung werden mit Hilfe von Booleanvariablen innerhalb der üblichen Methoden Informationen an den *Server*, bzw. den *Client* weiter gegeben.

1.2 MapLoader

Über den *MapLoader* werden die Spielfelder aus .txt-Dateien ausgelesen (siehe Kapitel 3). In einem Integerarray verpackt, wird dann über Getter- und Setter-Methoden die Kommunikation zwischen dem MapLoader und anderen Klassen geregelt. Verwendende Klassen sind hier das Menue, sowie die Map, der MapEditor und die Bombe.
Die Möglichkeit des Speicherns und Ladens der Level wird auch über die MapLoader-Klasse geregelt. Hier wird mit den *java.io.File**-Bibliotheken des JDK eine Schnittstelle zwischen dem Programm und den .txt-Dateien der Level hergestellt (siehe Kapitel 3 und 4).

1.3 Map

erbt von JPanel

Das *Map*-Objekt verwaltet Positionen und dient zur Kollisionsabfrage. Erzeugt wird es im *Menue*, wo auch die Getter- und Setter-Methoden zum Zugriff auf die Objektmethoden zu finden sind. Zudem wird im Konstruktor ein Integerarray bomb erzeugt, in dem in Matrizenform die Bomben gespeichert werden. Genutzt werden diese von der *Bombe* und der *Zeit*.
Des Weiteren werden hier über das *Menue* Veränderungen im Spielfeld registriert und anschließend entsprechende Informationen an die GUI-Verwaltung des *Menue*s weiter gegeben.

1.4 Bombe

erbt von JLabel

In der *Bombe* werden die Timer initialisiert, d.h. *Zeit*-Objekte erzeugt, welche als Threads im Hintergrund des Programms ablaufen und die Explosion der Bombe auf dem Spielfeld auslösen. Das Objekt wird direkt auf das Spielfeld abgebildet und für Kollisions- oder Ereignisabfragen genutzt. Auch die Aktionsberechnung findet in der Methode bombe_detoniert() statt (siehe Kapitel 2).

1.5 Zeit

erbt von JLabel

Das *Zeit*-Objekt wird von der Bombe initialisiert und steuert, wie bereits in 1.4 beschrieben, die Explosion der Bombe. Die Kollisionsabfragen werden geprüft und der Timer kann, je nach Ereignis frühzeitig beendet werden. Auch die Eigenschaften der Explosion (speziell die Ausdehnung im Spielfeld) werden mit Hilfe des get_game()-Getters des *Menue*s bestimmt.
In einer private-Klasse *Schwierigkeit* (erbt von java.util.TimerTask) wird der Timer zum regulieren der Schwierigkeit durch begrenzte Zeit zum Erreichen des Zieles verwaltet.

1.6 Hulk

Das *Hulk*-Objekt wird im *Menue* erzeugt und dient zum Verwalten der Spielereigenschaften, das heißt hier werden die verwendbare Bombenanzahl und der aktuelle Explosionsradius gespeichert und von dort auch in jede Methode, welche mit der Position des Spielers oder den anderen genannten Eigenschaften arbeitet, übergeben.

1.7 MapEditor

erbt von JPanel

Der MapEditor wird als eigenes Fenster im Frame abgebildet. Die Klasse besteht im Wesentlichen aus Methoden, welche über *JRadioButton*s mit dem Benutzer interagiert (siehe Kapitel 3). In einem Array werden die Eingaben gespeichert und mit Hilfe von Konsistenzprüfung, das heißt der Prüfung auf ungültige Eingaben wie zwei erste Spieler, die Gültigkeit des Spielfeldes sichergestellt. Dies erfordert eine Reihe von Gültigkeitsabfragen parallel zur Laufzeit, welche nach jeder Eingabe durchgeführt werden.

1.8 Server

erbt von Thread

Die *Server*-Klasse importiert Elemente der java.io.*- und java.net.*-Bibliotheken. Sie wird als Grundlage für die Netzwerkkommunikation des Programms genutzt und legt die IP-Adresse, welche der *Client* zum Verbindungsaufbau benötigt, fest. Nach dem Initialisieren des Spieles wird auf die Antwort des *Client* gewartet. Die Methode warte() wird zudem auch für andere Netzwerkabfragen wie ein Spielneustart verwendet. Die Kommunikation wird per TCP/IP durchgeführt und ist textbasiert, es werden also Strings verschickt, empfangen und ausgewertet. Bei Verbindungsabbruch oder fehlender *Client*-Antwort werden ExceptionHandlings in der Klasse genutzt. Beendet wird der Thread, indem durch `lan_modus_beenden()` über das *Menue* ein Abbruchsignal an die starten()- und die warten()-Methode geschickt wird.
Für den regulären Spielablauf nutzt die Klasse die Schnittstellen zum *Menue,* der *Map*, des *MapLoader*s etc.

1.9 Client

erbt von Thread

Wie auch der *Server* verwendet der *Client* Strings für die Netzwerkkommunikation und läuft parallel zur Laufzeit des Programmes. Über die Methode starten() wird der *Client* von run() (aus Thread) gestartet und der Thread gesteuert. Beendet wird dieser, wenn die starten()- oder die warten()-Methode abgebrochen wird (siehe 1.8 Server).

1.10 Sound

Die *Sound*-Klasse dient lediglich der Übersicht und der Verwaltung der Soundeffekte. Hier stehen die Pfade, über die auf die .wav-Dateien (siehe Kapitel 4) zugegriffen wird. Die Methode StdAudio.play() wird aus der StdAudio-Klasse genutzt.

1.11 StdAudio

In diesem Programm wird von der *StdAudio*-Klasse lediglich die play()-Methoden genutzt, welche über die Applet-Funktion die Tondateien als Threads abspielt.

2 Aufbau/Grundlagen des Programms

2.1 Das Spielfeld

Auf dem Spielfeld werden alle grafischen Schritte wie Bewegung, Explosionen, Blöcke usw. dargestellt. Es besteht aus verschiedenen 13x13 Arrays. Das Spielfeld besteht zudem aus drei Ebenen. Die erste Ebene enthält die Labels, also die Grafiken, die in der GUI angezeigt werden. Diese einzelnen Labels werden mit Icons beladen. Diese Icons bestehen sowohl aus png-Dateien, als auch aus animierten gif-Dateien (siehe Kapitel 4). Die zweite Ebene speichert die Bewegungen und Aktionen und stellt sie für Abfragen bereit. Die dritte Ebene ist ein *Bombe*-Array, welches mit *Bombe*-Objekten gefüllt ist und die Explosionen steuert. In allen drei Ebenen werden Kollisionsabfragen und Synchronisationen durchgeführt.

2.2 Bewegungen

Bei der Bewegungsmethode MoveHulk aus *Map* wird mit einer Kollisionsabfrage gearbeitet (2. Ebene). Hier wird überprüft, ob neben der Spielfigur ein Element positioniert ist. Wird neben der Figur eine Mauer platziert, so kann diese nicht passiert werden. Bewegt sich der Spieler über den Weg, so vertauschen sich die Postionen des Wegs und des Spielers auf der 1. und 2. Ebene.

2.3 Ereignisse

Eine ähnliche Kollisionsabfrage wie in Kapitel 2.2 wird bei der Explosion verwendet. Hier wird zuerst der Radius der Explosion überprüft, um dann die Felder die im Radius liegen mit dem Bild der Explosion zu überschrieben (1. und 3. Ebene). Nach Ablauf des Explosionstimers, welcher ein *Zeit*-Objekt ist, werden die Explosionsbilder wieder mit dem Weg überschrieben. Außerdem werden Blöcke die innerhalb des Explosionsradius liegen „zerstört“ und mit einem Wegicon überschrieben (Ebene 1 und 2). Wurde hinter einem Block ein Item oder ein Ausgang versteckt, erscheinen diese zuerst und werden erst nach Aufnahme durch den Spieler durch ein Wegfeld ersetzt. Die Veränderung an der Spielereigenschaft wird in dem entsprechenden *Hulk*-Objekt gespeichert und auch beim Legen einer Bombe dort abgerufen.

3 Benutzerschnittstellen

3.1 Die GUI

Zum Erzeugen der GUI von BomberHulk werden die java.swing.*- und java.awt.*-Bibliotheken verwendet. Es gibt eine Menüleiste, welche sich oberhalb des Spielfeldes befindet. Hier kann man über verschiedene Buttons auf die Funktionen des Spieles zugreifen, wie z.B. Schwierigkeitsgrad

oder Levelwahl. Außerdem kann man hier auch den Map-Editor, sowie den Netzwerk- oder Hot Seat Modus auswählen und starten. Die GUI ermöglicht es dem Spieler ohne Texteingaben gewisse Programmstellen auszuführen ohne sie in der Konsole ausführen zu müssen. Außerdem wird dem Spieler durch die GUI angezeigt wie viele Bomben er hintereinander legen darf und wie hoch sein Bombenradius ist. Dieser Vorgang geschieht im unteren Teil des Spielfeldes. Ergänzend dazu ist noch ein Ablauftimer sichtbar, welcher die verbleibende Spielzeit in Abhängikeit zum Schwierigkeitsgrad anzeigt. Wie auch beim Spielfeld selbst, wird hier mit JLabels gearbeitet, welche mit den einzelnen Funktionen beladen sind. Das Nutzen der Bibliotheken geschieht mit Hilfe von AbstractAction-Klassen innerhalb der *Menue*-Klasse. Hier werden die entsprechenden Aktionen ausgewertet und weitere Methoden aufgerufen.

3.2 Der Map-Editor

Der Map-Editor hat die Funktion, dem Spieler die Möglichkeit zu geben ein eigenes Level zu erstellen. Im Map-Editor wird zunächst eine „leere“ Map erstellt, welche standardmäßig aus einem Rand aus Mauern und Wegen besteht. Hier wird genau wie beim Spielfeld (Kapitel 2.1), mit JLabels gearbeitet. Man kann die einzelenen Elemente der Map auswählen und diese bliebig auf der Map platzieren. Will man diese Map speichern, so wird eine txt-Datei erstellt. In dieser Datei werden die Iconnummer gespeichert und als Matrixform aufgelistet. Im der Klasse MapLoader wird diese txt-Datei dann verarbeitet und die erstellt Karte kann geladen werden (siehe Kapitel 3.3).

3.3 Laden und Speichern

Das Laden einer Karte geschieht durch Bibliotheksmethoden zum Einlesen von Char-Elementen aus einer Textdatei mit Hilfe der Charakter.getNumericValue()-Methode der *java.io.FileReader*-Bibliothek. Die Elemente, welche in der Textdatei verwendet werden, werden in Kapitel 4 beschrieben. Die Char-Elemente werden von Methoden der *MapLoader*-Klasse interpretiert und verarbeitet. Letztendlich werden die interpretierten Zeichen in einem Integerarray gespeichert und an die *Map*-Klasse weiter gegeben.
Das Speichern funktioniert ähnlich dem Laden, in dem mit Hilfe der *java.io.FileWriter*-Bibliothek das entsprechende Array in eine Textdatei ausgelesen wird.

4 Datenformate

4.1 Sounds (.wav)

Im Programm werden verschiedene Datenformate verwendet und ins Programm eingebunden. Um den Sound im Spiel zu erzeugen werden .wav Dateien verwendet, welche in die Methode play() von *StdAudio* eingebunden werden. Sie müssen im src-Ordner vorhanden sein, damit die Sounds z.B. beim Explodieren der Bombe auftreten.

4.2 Die Karte (.txt)

Die verschiedenen Level werden in txt-Dateien gespeichert und in der Klasse MapLoader verarbeitet. Die Maps werden als Array eingelesen und aus den verschiedenen Ziffern ergibt sich die Struktur der Map, desweitern wird zuerst der Iconsatz ausgelesen, dann die maximale Anzahl an Bomben und der Radius festgelegt. Zwischen den Werte wird der „:“ als Trennzeichen verwendet.
0 : Bomben-Item hinter Block versteckt
1 : Hulk
2 : Weg
3 : Block (zerstörbar)
4 : Mauer (nicht zerstörbar)
5 : Bombe
6 : Explosion
7 : Ausgang
8 : Ausgang hinter Block
9 : Flammen-Item hinter Block
10 : Spieler 2
11 : Explosion / Bomben-Item
12 : Bomben-item
13 : Explosion / Ausgang
14 : Explosion / Flammen-Item
15 : Flammen-Item

4.3 Bilder (.png, .gif)

Die Bilder die zur Erzeugung der Grafiken verwendet wurden sind im .png-Format. Kriterium zur Auswahl dieses Formates war, dass das Format von Java verwendbar sein sein muss. Außerdem ist das Format nicht so verlustbehaftet wie .jpeg. Für die animierten Teile des Spielfeldes wie die Bomben und Blöcke wurden .gif Dateien erstellt und ins Programm eingebunden.