Kapitel 20 Mehrere Level

Lernziele:
In diesem Kapitel gibt es keine neuen Inhalte

20.1 Strukturieren durch Methoden

Einige Methoden der Klasse SPIELSTEUERUNG sind mittlerweile recht lange geworden. Um die Lesbarkeit des Quelltextes zu erhöhen, kann man diesen mit Kommentaren strukturieren (Abbildung 1)

```
public SPIELSTEUERUNG()
   {
        // Attribute initialisieren
        punkteStand = 0;
        leben = 3;
        level = 1;
        punktewertNormalerKruemel = 1;
        punktewertPowerkruemel = 10;
        timer1 = 0;

        //Level vorbereiten
        labyrinth = new LABYRINTH();
        mampfi = new MAMPFI(labyrinth, labyrinth.MampfiStartXGeben(),
                                       labyrinth.MampfiStartYGeben());
        monsterliste = new MONSTER[4];
        monsterliste[0] = new MONSTER(labyrinth, 2, 4, "dunkelorange");
        monsterliste[1] = new MONSTER(labyrinth, 2, 4, "orange");
        monsterliste[2] = new MONSTER(labyrinth, 6, 4, "magenta");
        monsterliste[3] = new MONSTER(labyrinth, 6, 4, "dunkelmagenta");

        // Anzeige erzeugen und initiailiesien
        anzeige = new STEUERUNGSANZEIGE();
        anzeige.Anmelden(this);
        anzeige.SchriftFarbeSetzen("orange");
        anzeige.HintergrundFarbeSetzen("blau");
        anzeige.PunkteStandSetzen(punkteStand);
        anzeige.LebenSetzen(leben);
        anzeige.LevelSetzen(level);

        // Zufallsgenerator erzeugen
        zufallszahlGenerator = new Random();
   }
```

Abbildung 1: Konstruktor der Klasse SPIELSTEUERUNG

Eine weitere Möglichkeit zur Strukturierung ist, Anweisungsblöcke in eine Methode mit einem aussagekräftigen Namen auszulagern. Ein weiterer Vorteil dieser Vorgehensweise ist, dass die Methode auch an anderen Stellen verfügbar ist, ohne dass der Quelltext verdoppelt wurde. Letzteres hat große Nachteile hinsichtlich Wartungen und Ergänzungen.

Aufgabe 20.1
Erstelle in der Klasse SPIELSTEUERUNG eine Methode LevelVorbereiten, die die grau hinterlegten Anweisungen aus Abbildung 1 enthält.

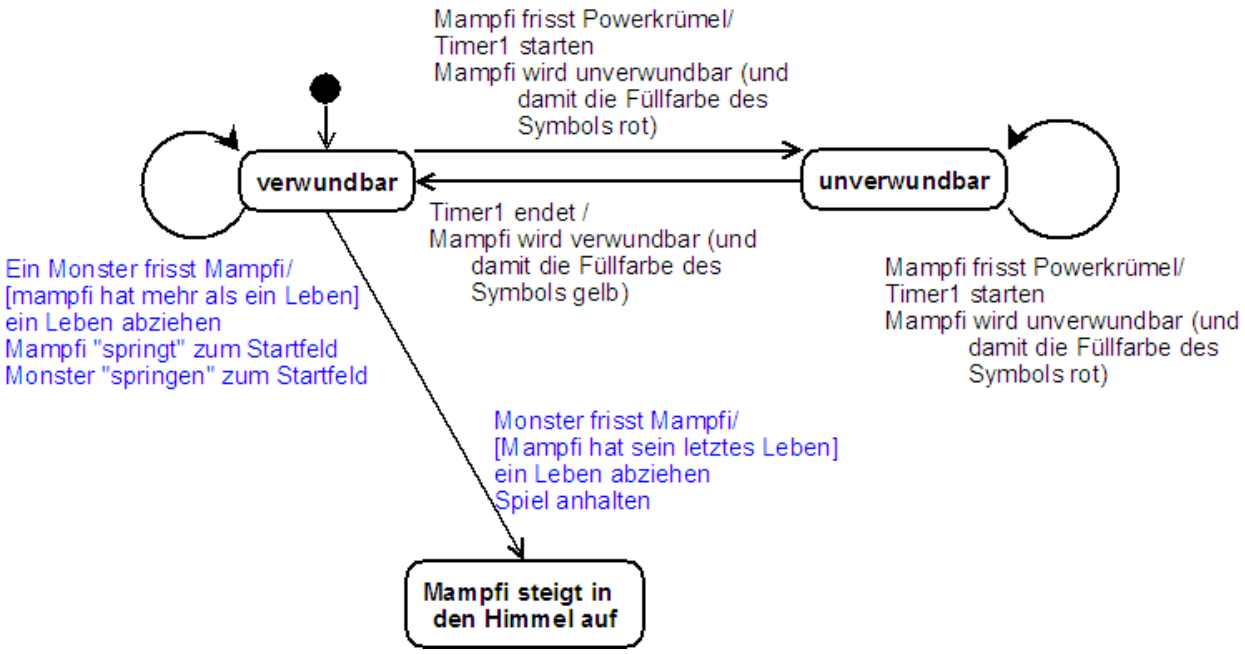
20.2 Planung über Zustandsdiagramme

Im letzten Kapitel war das einzig mögliche Spielende, dass Mampfi alle seine Leben verloren hat. Wie soll das Spiel weitergehen, wenn Mampfi alle Krümel gefressen hat?

Aufgabe 20.2

a) Ergänze Überlegungen zum Spielablauf mit verschiedenen Leveln in das Zustandsdiagramm aus Kapitel 19 Abbildung 10. Als Orientierung bzw. als Kopiervorlage ist es auf dieser Seite nochmals abgebildet.

b) Welche Ergänzungen an Attributen und Methoden ergeben sich aus Teilaufgabe a) für die Klasse SPIELSTEUERUNG. Notiere ein Klassendiagramm. Muss eine andere klasse auch noch ergänzt werden?



Eine Möglichkeit den Spielverlauf mit Levels festzulegen ist in der Abbildung 2 modelliert. Im Vergleich zu Kapitel 19 wurde ein Zustand und drei Zustandsübergänge hinzugefügt. Sie sind in der Abbildung farbig hervorgehoben.

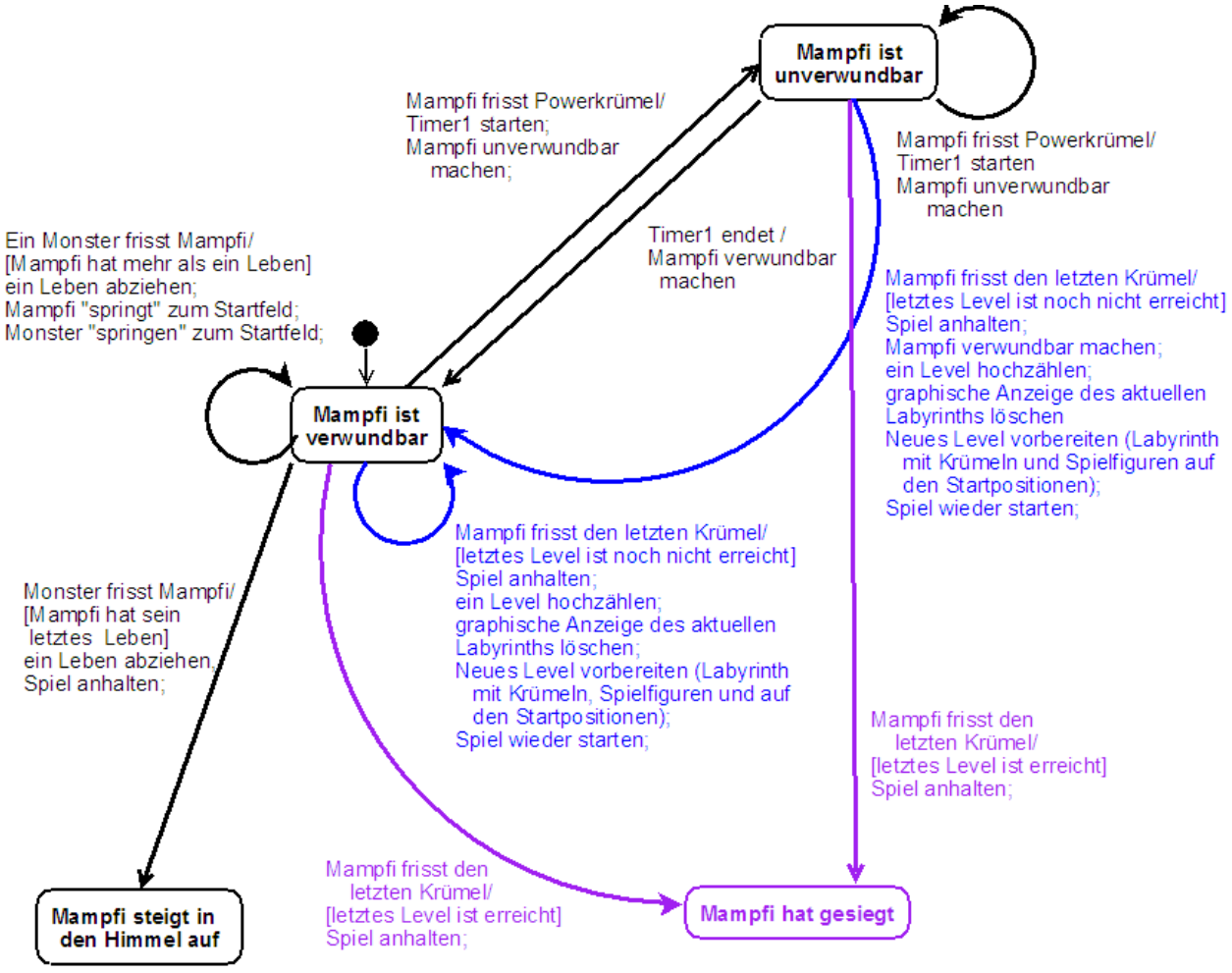


Abbildung 2: Zustandsdiagramm zur Planung des Spielablaufs, der in der Klasse SPIELSTEUERUNG gesteuert wird. Die Ergänzungen im Vergleich zu Kapitel 19 sind farblich hervorgehoben.

Aus diesem Zustandsdiagramm ergibt sich Änderungsbedarf für die Klassen SPIELSTEURUNG und LABYRINTH. Alle notwendigen Änderungen lassen sich aus der Beschriftung der neuen Zustandsübergänge ablesen.

i) auslösendes Ereignis: „Mampfi frisst den letzten Krümel“
Es ist sinnvoll eine Methode mit einem Rückgabewert vom Typ boolean zu ergänzen, die überprüft, ob der letzte Krümel gefressen wurde. Da die Krümel im Labyrinth liegen, sollte die Methode Teil der Klasse LABYRINTH sein.
--> Ergänze in der Klasse LABYRINTH eine Methode mit dem Kopf *boolean SindAlleKruemelGefressen()*

ii) Bedingung: „Letztes Level ist noch nicht erreicht“ bzw. „Letztes Level ist erreicht“
Wie viele Level hat das Spiel? Dies muss festgelegt werden, um eine entsprechende Überprüfung vornehmen zu können. Deshalb muss in der SPIELSTEUERUNG ein Attribut *levelAnzahl* vom Typ int ergänzt werden.

iii) Ausgelöste Aktionen, die (im Wesentlichen) bereits implementiert

„Spiel anhalten“

Dies gibt es schon in Form des Aufrufs

```
steuerungsanzeige.TickerAnhalten()
```

„Ein Level hochzählen“

Dies ist eine einfache Zuweisung.

„Spiel wieder starten“

Dies gibt es schon in Form des Aufrufs

```
steuerungsanzeige.TickerStarten(200)
```

„Mampfi verwundbar machen“

Dies gibt es schon in Form des Aufrufs

```
mampfi.VerwundbarSetzen(true)
```

„Neues Level vorbereiten"

Dies wird erreicht durch die in Kapitel 20.1 verfasste Methode *LevelVorbereiten()*

iv) Ausgelöste Aktionen, die neu sind:

„Graphische Anzeige des aktuellen Labyrinths löschen“

Hier gibt es Hilfe durch das Backend. Mit der Methode *LabyrinthAnzeigeLoeschen()* der Klasse STEUERUNGSANZEIGE wird im übertragenen Sinne die Tafel gewischt.

Ob das auslösende Ereignis aus i) erreicht wurde, muss nach jedem Spielzug überprüft werden. So müssen alle Punkte aus ii) und bis iv) in passender Struktur in der Methode *SpielzugAuswerten* der Klasse SPIELSTEUERUNG ergänzt werden.

Aufgabe 20.3

a) Ergänze in der Klasse LABYRINTH die Methode *SindAlleKruemelGefressen*. Du kannst die Methode testen, indem du ähnlich zu Aufgabe 19.9, indem du (vorübergehend) passende Methodenaufrufe über die Tastatur ermöglichst. Z. B. kannst du das Spiel mit 'P' anhalten und der Taste 'T (Nummer 84) den Methodenaufruf

```
System.out.println(labyrinth.SindAlleKruemelGefressen());
```

zuordnen. Auf der Konsole erhältst du dann die Rückmeldung true bzw. false.

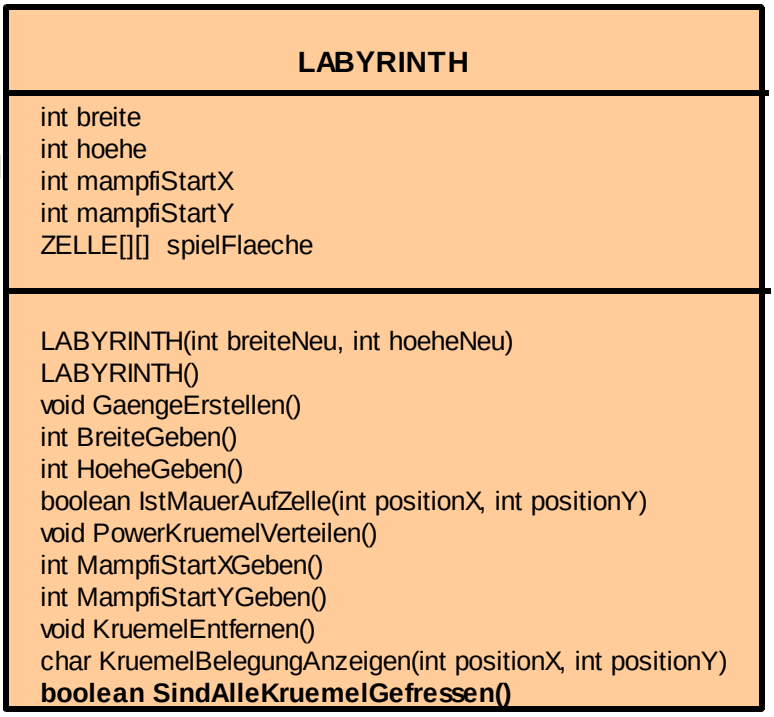
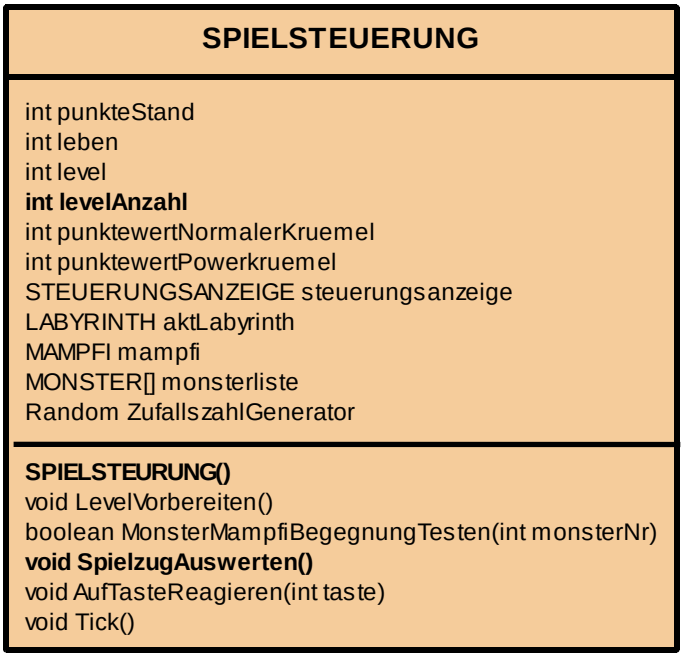


Abbildung 3: Ergänzung der Methode SindAlleKruemelGefressen in der Klasse LABYRINTH

b) Ergänze entsprechend den Überlegungen oben die Klasse Spielsteuerung. Das Klassen-diagramm in Abbildung 4 gibt dir einen Überblick, an welchen Stellen du Änderungen vornehmen musst.

Solltest du Hilfe benötigen findest du Tipps im Anhang.

Abbildung 4:
Aktueller Stand der Klasse SPIELSTEUERUNG



20.3 Vorläufiges Ende

An dieser Stelle endet (vorerst) das Projekt Krümel und Monster. Du hast ein Spiel selbst programmiert und dabei vermutlich festgestellt, dass dahinter viel Arbeit steckt. Wenn du Interesse hast, kannst du das Spiel an vielen Stellen weiterentwickeln. Ein paar Ideen seien hier genannt, sicher fallen dir noch mehr ein:

- Erhöhung der Geschwindigkeit aller Spielfiguren mit zunehmenden Level.
- Erhöhung der Geschwindigkeit der Monster im Vergleich zu der von Mampfi mit zunehmenden Level.
- Der Spieler hat einmalig im Spiel die Möglichkeit eine Mauer, die vor Mampfi steht zu entfernen.
- Die Monster sollten intelligenter werden und sich nicht nur zufällig im Labyrinth bewegen.
- Es passiert, dass Mampfi durch ein Monster "hindurchgehen" kann, ohne gefressen zu werden. Das ist kein Graphikfehler! Überlege dir die Ursache und verbessere dein Spiel.
- ...

Ich hoffe du hattest Spaß an dem Projekt. Solltest du ein paar neue Ideen gefunden und in deinem BlueJ-Projekt umgesetzt haben, würde ich mich über eine Zusendung unter brichzin@tcs.ifi.lmu.de freuen. (Idealerweise ist eine kleine Beschreibung dabei, was die Besonderheiten deines Spiels sind).

Und willst du dein Spiel jemand anderen zeigen, der kein BlueJ hat? Dann lies noch Kapitel 20.4

20.4 Krümel & Monster ohne BlueJ Spielen

Auf wenigen Rechnern ist BlueJ installiert, aber sehr viele Rechner können Java Programme ausführen. Wie lässt sich Krümel & Monster ohne BlueJ starten?

Im Moment beginnt das Spiel durch das Erzeugen eines Objekts der Klasse SPIELSTEUERUNG, d h. durch die Anweisung

```
new SPIELSTEUERUNG();
```

Ziel wäre es, dass dies automatisiert geschieht und nicht durch einen Mausklick innerhalb von BlueJ. Dies ist möglich, wenn es in einem Projekt mit Java Klassen die ausgezeichnete Methode mit dem Namen *main* gibt. Wird ein "Java-System" aufgerufen, dann überprüft es, ob es eine Methode main gibt. falls ja, wird diese aufgerufen. Steht nun innerhalb dieser main Methode die Anweisung

```
new SPIELSTEUERUNG();
```

ist das Ziel erreicht.

Aufgabe 20.4

a) Erstelle eine Klasse SPIEL, die kein einziges Attribut hat und nur den Konstruktor SPIEL als Methode. (Innerhalb des Konstruktors ist keine einzige Anweisung, da es ja auch kein Attribut gibt.)

b) Ergänze folgende Methode in der Klasse Spiel:

```
public static void main (String [] args)
    {
            new SPIELSTEUERUNG();
    }
```

c) Speichere das Projekt als jar Datei über das Menü "Projekt--> Als jar-Archiv speichern..." Gib im sich öffnenden Fenster die Kasse SPIEL als die Klasse an, in der sich die main-Methode befindet. Weiterhin ist es wichtig, das Backend als Benutzerbibliothek mitzuspeichern (Abbildung 5).

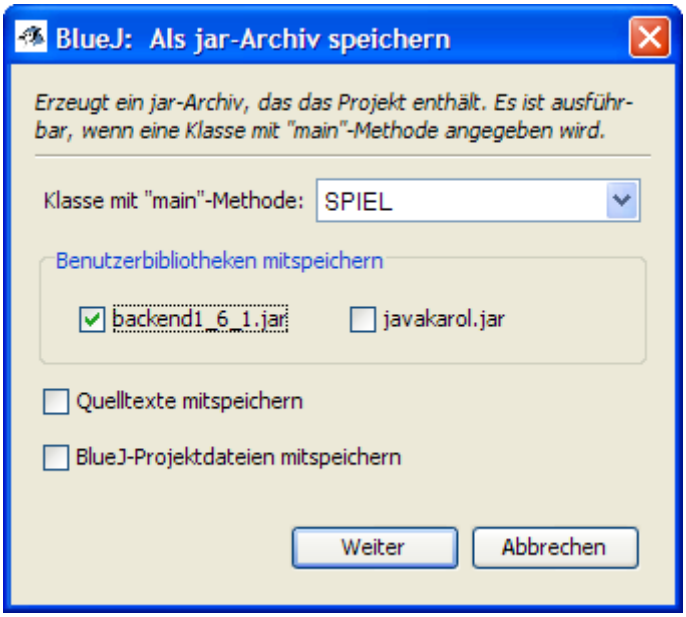


Abbildung 5: Auswahlfenster beim Erzeugen des jar-Archivs

d) Danach wirst du aufgefordert ein Verzeichnis auszuwählen in dem gespeichert wird und einen Dateinamen anzugeben.

e) BueJ erstellt nun ein Verzeichnis mit dem in d) angegebenen Namen. Dort befinden sich zwei jar Dateien: Das Backend und die ausführbare Datei deines Projekts.
Durch einen Doppelklick auf die Projekt-jar-Datei kannst du das Spiel auf jeden Rechner, auf dem Java installiert ist (ausreichend ist das Java Runtime Enviroment) starten.

20.5 Ausblicke

Im folgenden werden noch ein paar Ausblicke ohne ausführliche Anleitung gegeben:

Levelleditor

Ist einmal ein Anzeigefenster erstellt kann über das Menü "Labyrinth --> Editor starten" der Level-Editor gestartet werden(Abildung 6). Über ihn kann man einfach verschiedene Level, d.h. das Labyrinth mit Mauern und Powerkrümeln, Monster und Mampfi mit ihren Startpositionen erzeugen und abspeichern.

Abbildung 6:
Starten des Levleditors über das Menü des Anzeigefensters

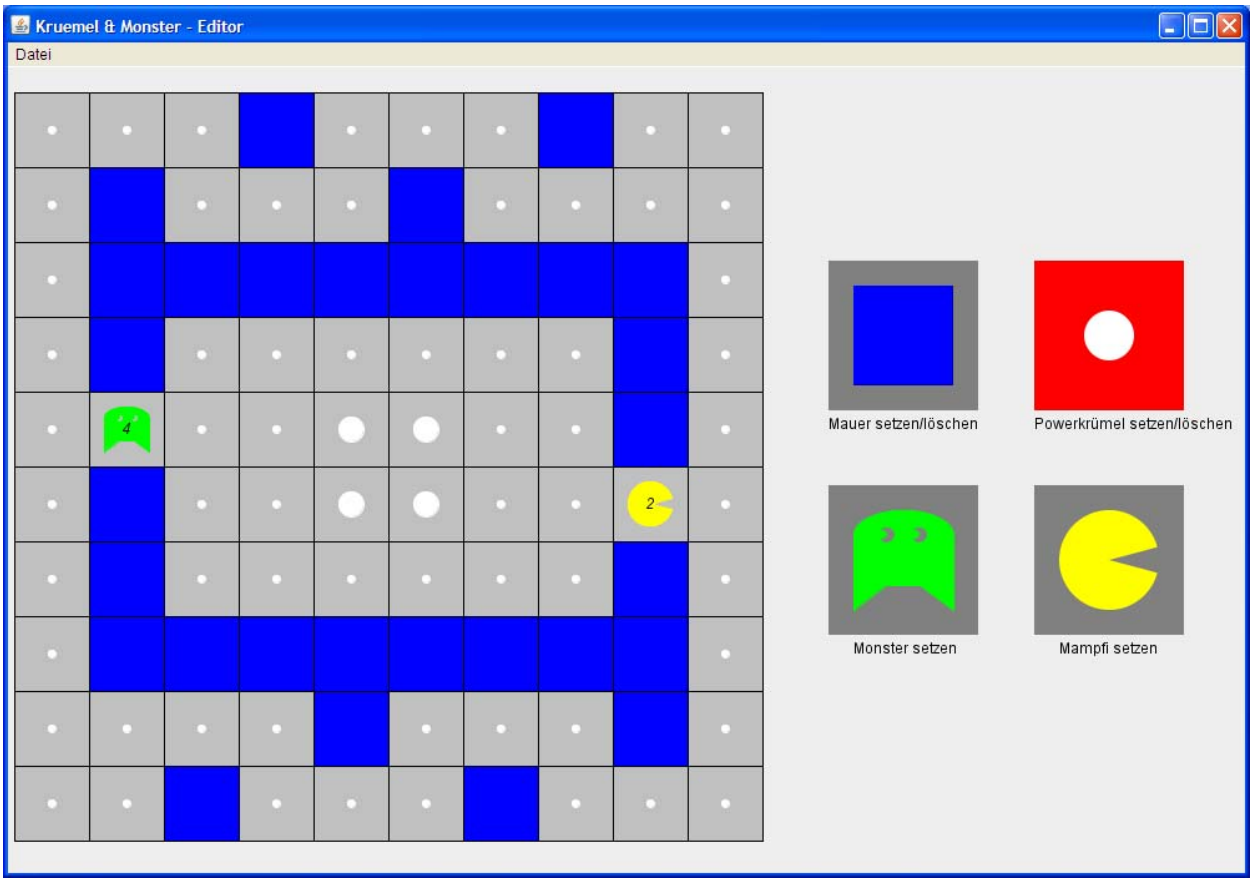


Abbildung 7: Levleditor

Die erzeugte Level-Datei kann man mit jedem Editor ansehen. Neben der Zellenanzahl und den Positionen von Mampfi, den Monstern und Krümeln wird pro Zelle gespeichert, ob sich dort ein Krümel ('K) oder eine Mauer ('M') befindet.

Dateileser

Als Hilfe zum Auslesen der gespeicherten Labyrinthdaten steht die Klasse DATEILESER zur Verfügung (Abbildung 8).

Hat die Klasse LABYRINTH eine Referenz auf ein Objekt Dateileser, so können für das Labyrinth relevante Daten wie die Startposition von Mampfi und der Monster, die Zelldaten ('K' für Krümel, 'M' für Mauer) ausgelesen werden.
Ein sinnvoller Ort für das Auslesen ist der Konstruktor, dort wird ja das Labyrinth erstellt.

Natürlich müssten in der Klasse LABYRINTH noch entsprechende Attribute zu den Startpositionen der Monster und der Powerkrümelpositionen ergänzt werden.

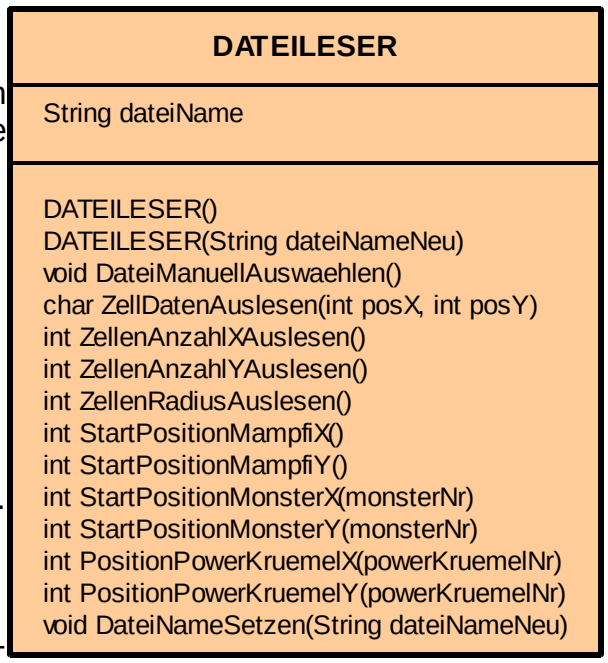


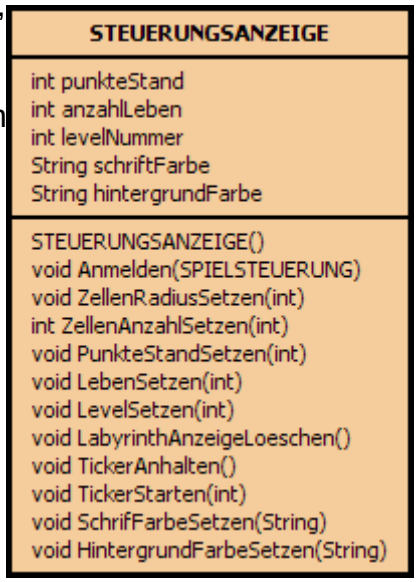
Abbildung 8:
Schnittstelle der Klasse DATEILESER

Labyrinthe mit mehr als 10 x 10 Zellenanzahl

Mit dem Level-Editor lassen sich auch Labyrinthe erzeugen, die mehr als 10 x 10 Zellen haben.
Wähle dazu das Menü "Datei --> Neu". Es erscheint ein Fenster, in dem die Seitenlänge des Labyrinths abgefragt wird (Abbildung 9).

Abbildung 9: Auswahl der Seitenlänge des Labyrinths über den Level-Editor

Um das Labyrinth dann auch im Anzeigefenster darstellen zu können, bedarf es einiger Überlegungen bzw. Umstellungen:
Der Bereich für das Labyrinth im Anzeigefenster hat konstant 600 Pixel x 600 Pixel. Sollen nun mehr als 10 x 10 Zellen dargestellt werden, muss
a) die Zellgröße verändert werden und
b) weiterhin die Größe aller Symbole z.B. das Symbol von Mampfi angepasst werden.



zu a)
Die Grundeinstellung erfolgt über die Methoden

```
int ZellenAnzahlSetzen(int anzahl)
```

und

```
void ZellenRadiusSetzen(int
                        zellenRadiusNeu)
```

der Klasse STEUERUNGSANZEIGE (Abbildung 10).
Die Maße des Zellensymbols werden dadurch automatisch gesetzt.

Abbildung 10:
Schnittstelle der Klasse STEUERUNGSANZEIGE

zu b)
Ein Vergleich der Größen der Symbolobjekte zwischen einem 10 x 10 und 20 x 20 Labyrinth soll die prinzipielle Berechnung veranschaulichen:
Bei dem bisher verwendeten Feld mit 10 x 10 Zellen ergibt sich für jede Zelle ein Größe von 60 Pixel x 60 Pixel, wobei die Größe der Zellensymbol-Objekte automatisch gesetzt werden.:

Symbol für	Größe in Pixel	Bezug zur Zellengröße (ZG), wobei gilt ZG = 600 / Anzahl der Zellen pro Zeile
Zelle	Seitenlänge = 60 bzw. "Zellenradius" = 30 (60 ist der voreingestellte Wert; Man kann diesen über die Methode *ZellenRadiusSetzen* der Klasse STEUERUNGSANZEIGE ändern.)	ZG ZG/2
Mampfi	Radius = 30	ZG/2
Monster	keine Größe wählbar, wird automatisch vom Backend gesetzt	
Powerkrümel	15	ZG/4
Krümel	7	ca. ZG/8

Daraus ergibt sich für ein Feld mit 20 x 20 Zellen folgende Größen:

Symbol für	Bezug zur Zellengröße (ZG), wobei gilt ZG = 600 / 20 = 30	Größe in Pixel
Zelle	ZG ZG/2	Seitenlänge = 30 "Zellenradius" = 15
Mampfi	ZG/2	Radius = 15
Monster		keine Größe wählbar, wird automatisch vom Backend gesetzt
Powerkrümel	ZG/4	7
Krümel	ca. ZG/8	3

Bleibt noch als letzte Frage:
Wie erhält die Spielsteuerung aus der Level-Datei die Information über die Zellenanzahl?
Sie muss diese, wie auch das Labyrinth, über einen Dateileser aus der Level-Datei auslesen.

Anhang A: Tipps zu den Aufgaben

Tipps zu Aufgabe 20.3 a):

- Da alle Zellen überprüft werden müssen sind zwei ineinander geschachtelte Wiederholungen mit fester Anzahl nötig.
- Wenn eine Zelle keine Mauer ist und keinen Krümel hat, dann wird false zurückgegeben und die Methode beendet.
  Wurden alle Zellen überprüft, und ist nie der gerade beschriebene Fall eingetreten, wird true zrückgegeben.
- Sobald in einer Methode mit Rückgabewert eine return Anweisung ausgeführt wurde, wird diese beendet.

Noch nicht alles klar? Vielleicht helfen folgende Fragen

i) "Wenn eine Zelle keine Mauer ist" : Wie lässt sich diese Bedingung mit einem passenden Methodenaufruf formulieren?

ii) "Wenn eine Zelle keinen Krümel hat" : Wie lässt sich diese Bedingung mit einem passenden Methodenaufruf formulieren?

iii) Wie lässt sich aus i) und ii) eine einzige Bedingung formulieren?
(Wenn **eine Zelle keine Mauer ist und keinen Krümel hat**, dann ...).

Tipps zu Aufgabe 20.3 b):

- Deklaration des Attributs levelAnzahl.
  Initialisierung im Konstruktor nicht vergessen.
- Alle drei farbigen (d.h.neuen) Zustandsübergänge aus Abbildung 2 lassen sich in einem neuen Abschnitt der Methode SpielzugAuswerten zusammenfassen. Dazu sind zwei bedingte Anweisungen nötig:
  Die Bedingung
  ```
  mampfi.VerwundbarGeben() == false
  ```
  ermöglicht eine Unterschiedung zwischen den beiden blauen Zustandsübergängen.
  Die Bedingung
  ```
  level < levelAnzahl
  ```
  ist die Bedingung die schon aus dem Zustandsdiagramm ablesbar ist.

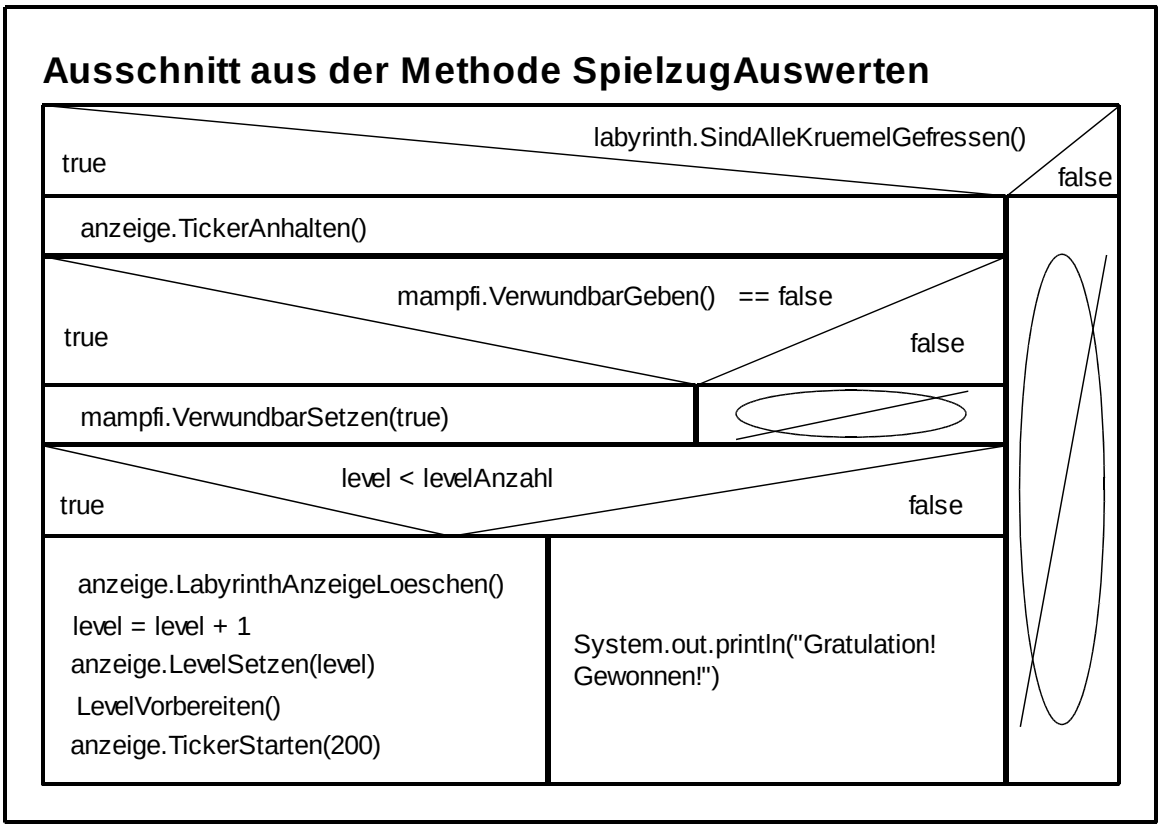
Noch mehr Tipps zu Aufgabe 20.3a):

i) "Wenn eine Zelle keine Mauer ist" :
```
if ( !(IstMauerAufZelle(zaehlerX, zaehlerY) )
```
oder
```
if ( IstMauerAufZelle(zaehlerX, zaehlerY) == false)
```

ii) "Wenn eine Zelle keinen Krümel hat"
```
if (KruemelBelegungAnzeigen(zaehlerX, zaehlerY) != 'L')
```

iii) Die Verknüpfung erfolgt mit dem Und-Operator &&
```
if ( !(IstMauerAufZelle(zaehlerX, zaehlerY)) &&
      (KruemelBelegungAnzeigen(zaehlerX, zaehlerY) != 'L'))
```

Noch mehr Tipps zu Aufgabe 20.3b):



Noch mehr Tipps zu Aufgabe 20.3a):