Kapitel 19 Fressen und gefressen werden

Lernziele:
In diesem Kapitel werden u.a. Zustandsdiagramme wiederholt und erweitert.
Neu sind logische Operatoren

19.1 Planung über Zustandsdiagramme

In Kapitel 15 wurde mit Hilfe eines Zustandsdiagramms die Methode *SpielzugAuswerten* geplant. Es haben sich daraus drei Fälle ergeben: Unterschiedliche Anweisungen waren nötig, je nachdem ob Mampfi eine Zelle mit normalen Krümel, mit Powerkrümel oder ohne Krümel erreicht. Abbildung 1 zeigt das Struktogramm der Methode *SpielzugAuswerten*.

Rumpf der Methode SpielzugAuswerten

labyrinth.KruemelBelegungAnzeigen(mampfi.PositionXGeben(), mampfi.PositionYGeben())		
'N' // normaler Kruemel	'P' // Powerkrümel	'L' // kein Krümel

punkteStand = punkteStand + punktewertNormalerKruemel
anzeige.PunkteStandSetzen(punkteStand)
labyrinth.KruemelEntfernen(positionX, positionY)
	mampfi.VerwundbarSetzen(false);
	punkteStand = punkteStand + punktewertPowerkruemel;
	anzeige.PunkteStandSetzen(punkteStand);
	labyrinth.KruemelEntfernen(positionX, positionY)
		// nichts tun

Abbildung 1: Struktogramm der Methode *SpielzugAuswerten*

Da es in Kapitel 15 noch keine Monster gab, sind diese bisher völlig unberücksichtigt. Dies muss sich ändern

Aufgabe 19.1
Erstelle ein Zustandsdiagramm, das die Zustandsänderungen **der Monster** im Laufe des Spiels beschreibt. Es müssen mindestens die Zustände (Monster ist) verwundbar und unverwundbar (= nicht verwundbar) vorkommen.

Eine sehr ausführliche Variante des gesuchten Zustandsdiagramms zeigt Abbildung 2.

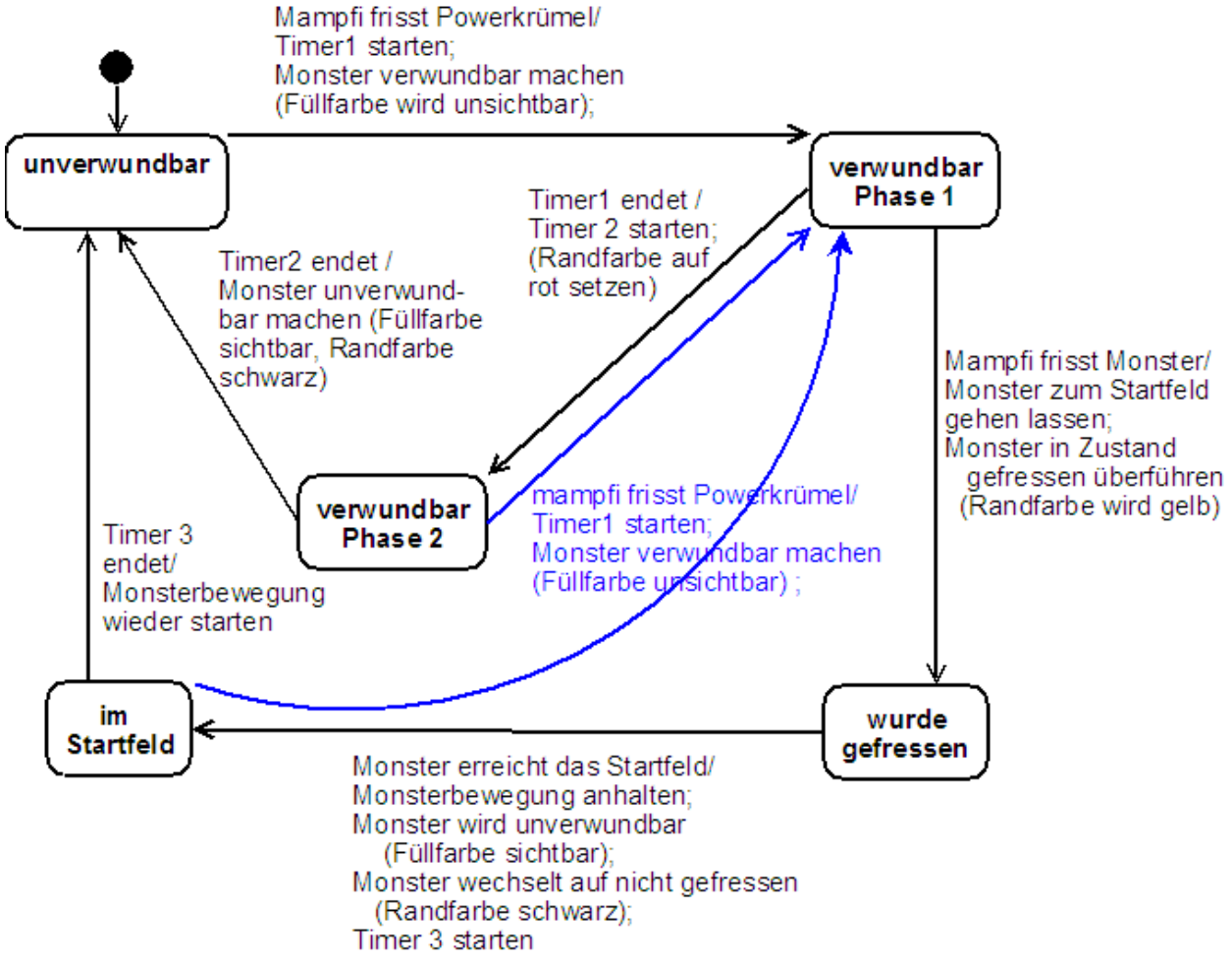


Abbildung 2:
Zustandsdiagramm zur Planung der Zustandsübergänge bei den Monstern – ausführliche Variante

Beachte die ausgelöste Aktion bzw. das auslösende Ereignis Timer1[1] startet bzw. Timer 1 endet.

Aufgabe 19.2

a) Erkläre deinem Banknachbarn knapp den Sinn dieses Timers in diesem Zusammenhang.

b) Sucht zusammen eine Möglichkeit, wie sich der Timer in dem aktuellen BlueJ-Projekt umsetzen lässt.
Hinweis: Es sind dazu keine neuen Klassen, keine neuen Methoden nötig.

Selbstverständlich werden Spiele durch mehr Zustände ihrer Spielfiguren interessanter. Für den Programmierer wird die Spielerstellung jedoch umfangreicher und komplizierter. Um nun in diesem Kapitel möglichst schnell eine spielbare Version von Krümel & Monster zu erstellen, wird eine Umsetzung einer deutlich einfacheren Variante entsprechend dem Zustandsdiagramm in Abbildung 3 erarbeitet.

1 timer: engl. Zeitmesser (für ein bestimmtes Zeitintervall)

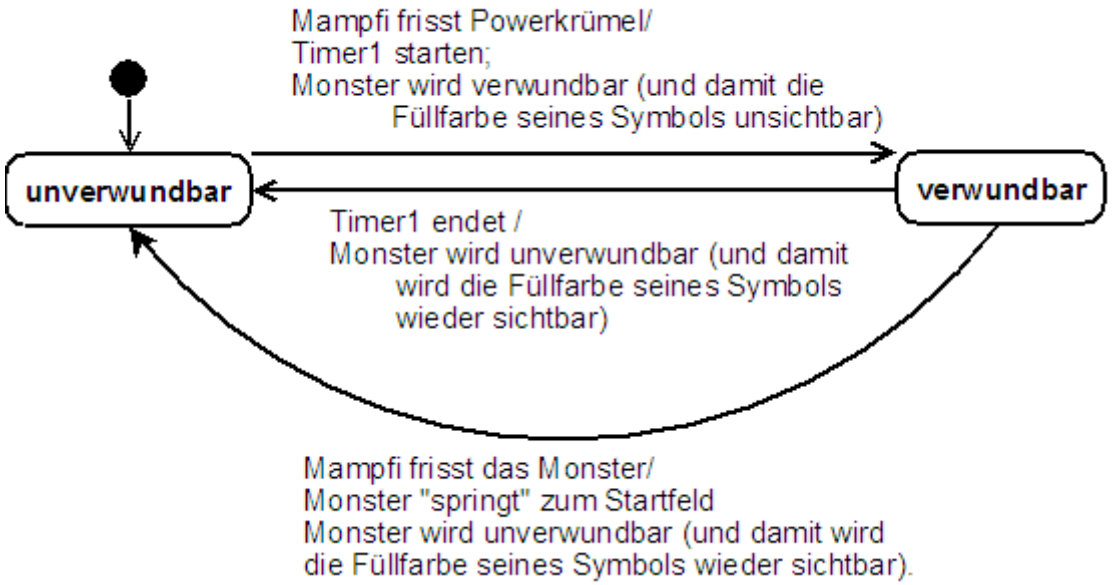
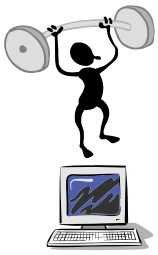


Abbildung 3:
Zustandsdiagramm zur Planung der Zustandsübergänge bei den Monstern – einfache Variante

Die Vereinfachungen sind dabei wie folgt:
- Die Monster werden ohne Vorwarnung nach Ablauf des Timer1 unverwundbar.
- Wird ein Monster gefressen, dann „springt“ es direkt auf sein Startfeld, ohne sich den Weg durch das Labyrinth zu suchen.
- Auf dem Startfeld verweilt das Monster nicht, sondern macht sofort wieder Jagd auf mampfi.

Als weitere Vereinfachung wird als Startfeld für alle Monster die Zelle (0/0) gewählt.



Aufgabe 19.3
Wenn du es dir zutraust, dann setze das Zustandsdiagramm in Abbildung 3 eigenständig um. Wenn du damit fertig bist und ausreichend getestet hast, kannst du die Features deines Programms erweitern. Plane dein Vorgehen in Form von Zustandsdiagrammen bzw. ergänzten Klassendiagrammen.
Bist du fertig, dann fahre fort im Kapitel 19.3.
Fühlst du dich unsicher, dann findest du im Folgenden eine schrittweise Anleitung.

Aufgabe 19.4
Analysiere das Zustandsdiagramm in Abbildung 3. Welche der auslösenden Ereignisse können bereits als Bedingungen abgefragt werden, welche nicht.
Welche Methodenaufrufe stehen für die ausgelösten Aktionen. Überlege auch hier, welche Methoden schon vorhanden sind und welche nicht.

Die Analyse des Zustandsdiagramms aus Abbildung 3 ergibt folgendes:

a) Zustandsübergang von unverwundbar nach verwundbar

auslösendes Ereignis "Mampfi frisst Powerkrümel":

Der Rückgabewert des Aufrufs der Methode

```
labyrinth.KruemelBelegungAnzeigen(mampfi.PositionXGeben(),
 mampfi.PositionYGeben())
```

ist 'P'. Diese Bedingung wird bereits in der **Methode *SpielzugAuswerten*** der Klasse SPIELSTEUERUNG abgefragt. **Es ist daher sinnvoll die ausgelösten Aktionen genau an dieser Stelle zu ergänzen.**

ausgelöste Aktion "Timer 1 Starten":

Entsprechend den Überlegungen zum Timer aus Aufgabe 19.2 (Lösung siehe Anhang), muss das Attribut timer1 z.B. auf den Wert 30 gesetzt werden.

ausgelöste Aktion Monster wird verwundbar:

```
monsterliste[zaehler].VerwundbarSetzen(true);
```

Dieser Methodenaufruf muss für alle Monster erfolgen.

b) Zustandsübergang von verwundbar nach unverwundbar

b1) auslösendes Ereignis "Timer 1 endet":

Entsprechend den Überlegungen zum Timer aus Aufgabe 19.2 (Lösung siehe Anhang), erreicht das Attribut timer1 den Wert 0. An jeder Stelle in der Klasse SPIELSTEUERUNG kann die Bedingung `timer1 == 0` überprüft werden. Das schrittweise Verkleinern des Werts des Attributs timer1 findet in der **Methode *Tick*** der Klasse SPIELSTEUERUNG statt. **Es ist daher sinnvoll die ausgelösten Aktionen genau an dieser Stelle zu ergänzen.** (Vgl. auch Lösung zu Aufgabe 19.2)

ausgelöste Aktion "Timer 1 Starten":

Entsprechend den Überlegungen zum Timer aus Aufgabe 19.2 (Lösung siehe Anhang), muss das Attribut timer1 z.B. auf den Wert 30 gesetzt werden.

ausgelöste Aktion Monster wird unverwundbar:

```
monsterliste[zaehler].VerwundbarSetzen(false);
```

Dieser Methodenaufruf muss für alle Monster erfolgen.

b2) auslösendes Ereignis "Mampfi frisst Monster":

Dieses Ereignis kann über aktuell existierende Methoden und Attribute nicht überprüft werden.

ausgelöste Aktion "Monster springt zum Startfeld":

Die könnte man über mehrere Zuweisungen und Methodenaufrufe umsetzen. Es wäre jedoch sinnvoll eine eigene Methode zu schreiben, die dies bündelt.

ausgelöste Aktion Monster wird unverwundbar:

```
monsterliste[zaehler].VerwundbarSetzen(false);
```

Dieser Methodenaufruf muss für alle Monster erfolgen.

Aufgabe 19.5

Setze die Punkte a) und b1) entsprechend den Überlegungen oben in deinem BlueJ-Projekt um. Teste die Änderungen.

Beachte:

Das Zustandsdiagramm in Abbildung 3 betrachtet nur die Übergänge der Monster! Berücksichtigt ist nicht, dass Mampfi beim Ablauf des Timers wieder verwundbar werden muss.

19.2 Fressen und ...

Entsprechend der Überlegungen in 19.1 sind folgende zwei Punkte noch nicht verfügbar:

i) Mampfi frisst Monster
ii) Monster springt zum Startfeld

Aufgabe 19.6

Überlege, wie sich die Lücken schließen lassen. Bestandteil deiner Überlegungen sollte sein,

a) ob und wenn ja in welchen Klassen Attribute und Methoden hinzugefügt werden müssen (erweitertes Klassendiagramm).
b) ob und wenn ja in welchen Klassen Methoden verändert werden müssen.
c) ob und wenn ja zwischen wen eine Objektkommunikation notwendig ist (Sequenzdiagramm).

zu i) Mampfi frisst Monster

- Hier muss zunächst erkannt werden, dass sich mampfi und ein Monster auf der gleichen Zelle befinden. Für diese Aufgabe sollte eine Methode *MonsterMampfiBegegnungTesten* in der Klasse SPIELSTEUERUNG ergänzt werden. Sie hat als Eingabewert eine Monsternummer vom Typ int – Es wird die Begegnung von mampfi mit dem entsprechenden Monster überprüft. Und als Ausgabewert true, falls die beiden auf der gleichen Zelle stehen, und false falls nicht.
  Hinweis: Diese Methode kann auch für den später zu erledigenden Punkt „Monster frisst Mampfi“ verwendet werden.

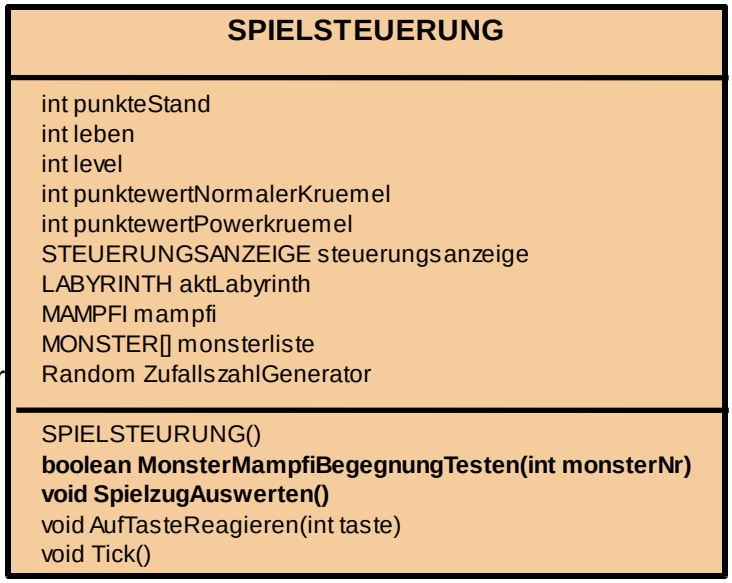


Abbildung 5: erweitertes Klassendiagramm der Klasse SPIELSTEUERUNG - Ergänzungen bzw. Änderungen sind fett hervorgehoben

- Die Begegnung muss für alle vier Monster überprüft werden.
  Dies ist eine Ergänzung in der Methode *SpielzugAuswerten* der Klasse SPIELSTEUERUNG.
- Findet eine Begegnung statt, so sollten werden dem Spieler z. B. 200 Punkte gut geschrieben werden. Auch dies ist eine Ergänzung in der Methode *SpielzugAuswerten* der Klasse SPIELSTEUERUNG..
  Beachte: Für die letzten beiden Spiegelstriche ist entscheidend, dass Mampfi unverwundbar ist. Nur dann dürfen die beschriebenen Anweisungen ausgeführt werden. Ist Mampfi jedoch verwundbar, dann sehen die Konsequenzen einer Begegnung anders aus (s. Kap. 19.3).

Aufgabe 19.7

Überlege dir den Methodenrumpf der Methode *MonsterMampfiBegegnungTesten.*

Damit eine Begegnung stattfindet, müssen die x- und y-Position von Mampfi und dem Monster übereinstimmen. Dies lässt sich über zwei ineinander geschachtelte bedingte Anweisungen überprüfen.

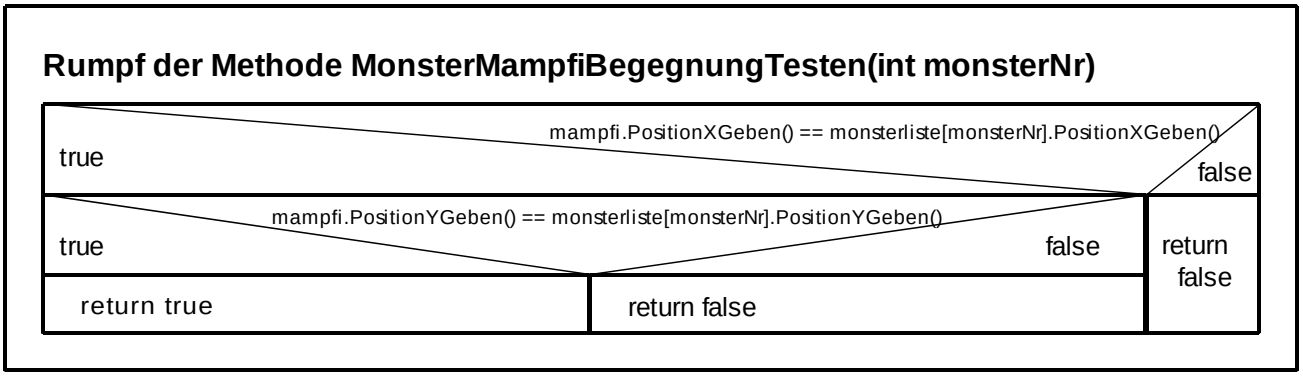


Abbildung 6: Umsetzung der Methode MonsterMampfiBegegnungTesten in der Variante mit einer geschachtelten bedingten Anweisung

Die Ausgabe true erfolgt nur, wenn müssen die x-Position von Mampfi und dem Monster übereinstimmen UND die y-Position von Mampfi und dem Monster übereinstimmen. Mit Hilfe von logischen Operatoren lässt sich dies kürzer und übersichtlicher notieren. Es reicht damit eine bedingten Anweisungen, eine Schachtelung ist nicht nötig.

Die logische Funktionen UND, ODER und NICHT kennst du von der Arbeit mit Rechenblättern oder Datenbankabfragen. In Java werden sie durch die Operatoren &&, || und ! umgesetzt. Damit ist eine Bedingung für die Formulierung des Methodenrumnpfs ausreichend (Abbildung 7).

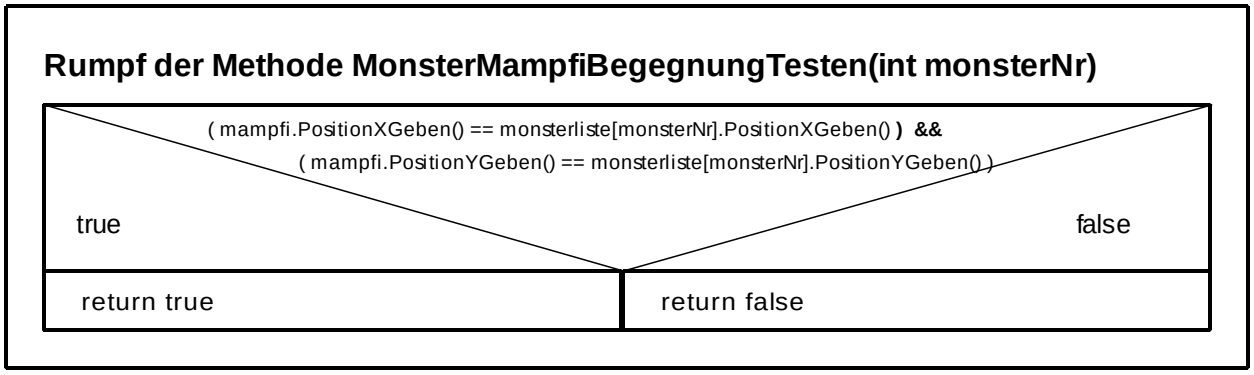


Abbildung 7: Umsetzung der Methode MonsterMampfiBegegnungTesten in der Variante mit logischen Operatoren

Aufgabe 19.8

Setze die Methode *MonsterMampfiBegegnungTesten* in der Klasse um.

Normalerweise ist bei den praktischen Aufgaben ergänzt: „Teste!“. Wie lässt sich diese Methode testen? Exakt zu dem Zeitpunkt, in dem sich Mampfi und ein Monster auf der selben Zelle befinden über das Kontextmenü des Objekts Spielsteuerung die Methode *MonsterMampfiBegegnungTesten* aufrufen. Das ist wohl nicht zu koordinieren. Man müsste das Spiel anhalten können!

Aufgabe 19.9

a) Die Klasse STEUERUNGSANZEIGE stellt eine Methode *TickerAnhalten()* zur Verfügung. Belege die Taste „P“ (Tastennummer 80) so, dass beim Drücken dieser Taste das Spiel angehalten wird.

b) Teste nun die Methode *MonsterMampfiBegegnungTesten,* indem du ein Objekt der Klasse SPIELSTEREUERUNG erzeugst, den Ticker (über die Taste „S“) startest, den Ticker in dem Moment anhältst (über die Taste „P“), wenn ein Monster auf dem gleichen Feld wie mampfi ist. Rufe in diesem angehaltenen Zustand über das Kontextmenü des Objekts Spielsteuerung die Methode *MonsterMampfiBegegnungTesten* auf und überprüfe, ob der Rückgabewert true ist. Prüfe auch den Rückgabewert für die anderen drei Monster.
(Das Spiel bzw. den Test kannst du über die Taste 'S' wieder fortsetzen.)

Aufgabe 19.10

Ergänze dazu die Methode *SpielzugAuswerten* der Klasse SPIELSTEUEURNG so, dass entsprechend der Überlegungen oben

a) Die Begegnung von Mampfi mit allen vier Monstern überprüft wird.

b) Bei einer Begegnung der Punktestand erhöht wird, jedoch nur, wenn Mampfi unverwundbar ist.
Hinweis: Um die Information, ob Mampfi unverwundbar ist zu erhalten, musst noch eine Methode in einer anderen Klasse ergänzt werden.

ii) Monster springt zum Startfeld
Hierzu muss eine Methode *AufStartpositionSpringen* oder allgemeiner *PositionSetzen* in der Klasse MONSTER ergänzt werden. Da Mampfi, falls er gefressen wird ebenso zum Startfeld springen muss, ist es sinnvoll diese Methode gleich in der Klasse SPIELFIGUR zu ergänzen.

Aufgabe 19.11

Ergänze die Methode *AufStartpositionSpringen* in der Klasse SPIELFIGUR. Verwende dafür vereinfachend die Zelle (0/0) einhheitlich als Startfeld bzw. eine andere Zelle auf der in deinem Labyrinth keine Mauer steht.
Ergänze an passender Stelle einen Aufruf der Methode *AufStartpositionSpringen* in der Klasse SPIELSTEUERUNG.

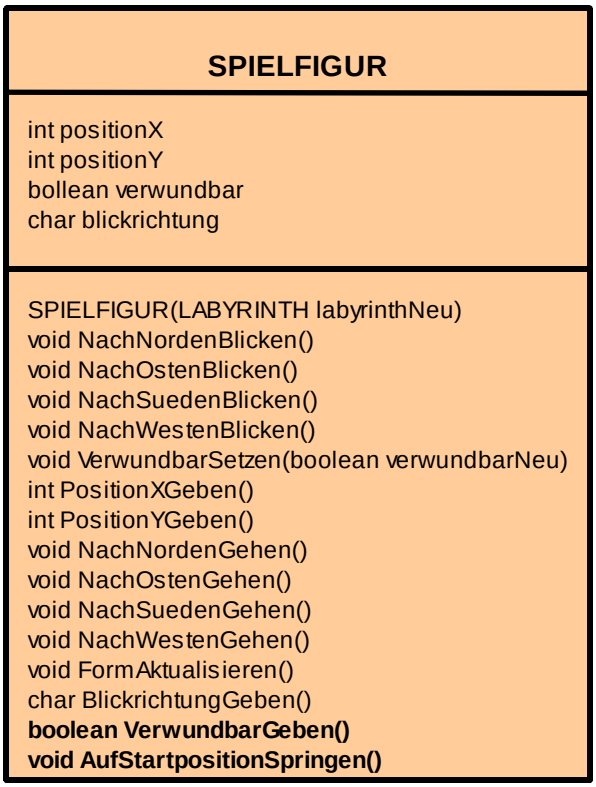


Abbildung 8: Neue Methode AufStartpositionSpringen in der Klasse SPIELFIGUR; die Methode VerwundbarGeben war bereits nötig für Aufgabe 19.10

19.4 ... gefressen werden

Das Zustandsdiagramm in Abbildung 3 modelliert die Übergänge hinsichtlich der Verwundbarkeit eines Monsters. Die Verwundbarkeit von Mampfi ist hier nicht berücksichtigt und könnten ergänzt werden. Übersichtlicher ist ist es jedoch ein (von den Monstern entkoppeltes) Zustandsdiagramm zu erstellen, dass nur die die Verwundbarkeit von Mampfi modelliert. Analog zu Abbildung 3 muss es wieder die Zustände verwundbar und unverwundbar geben, die sich jedoch auf Mampfi beziehen.

Aufgabe 19.12
Erstelle ein Zustandsdiagramm zu den eben beschriebenen Anforderungen. Reduziere dich wie bei den Monstern an dieser Stelle auf absolut notwendige Aktionen. Hast du weitere Ideen für „Besonderheiten“, so notiere sie und setze sie in einem späteren Schritt um.
Welcher dritte Zustand sollte sinnvollerweise ergänzt werden?
Beschreibe mit Worten, welche Bedingung erfüllt werden muss, dass ein Übergang in den dritten Zustand erfolgt.

Als weiterer Zustand muss das Spielende aufgenommen werden, denn wenn mampfi sein letztes Leben hat und gefressen wird, ist das Spiel zu Ende. Die Formulierung „wenn mampfi sein letztes Leben hat“ zeigt, dass der Zustandsübergang an eine Bedingung gekoppelt ist, nur wenn die Bedingung erfüllt ist (den Wert true hat) erfolgt der Übergang. Im Zustandsdiagramm gibt es die Möglichkeit dies darzustellen: Nach dem auslösenden Ereignis wird die Bedingung in eckigen Klammern notiert (Abbildung 9).

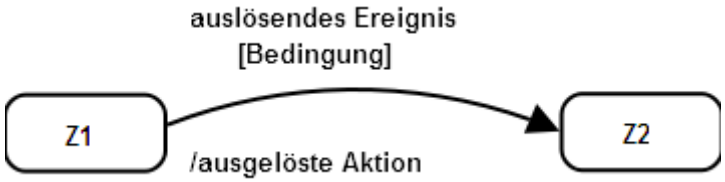


Abbildung 9: Zustandsübergang mit Bedingung: Ein Übergang von Zustand Z1 nach Zustand Z2 erfolgt beim Eintreten des auslösenden Ereignisses nur, wenn die Bedingung erfüllt ist.

Damit ergibt sich folgendes Zustandsdiagramm (Abbildung 10) für die Modellierung der Zustände von Mampfi im Kontext des „gefressen werdens“.

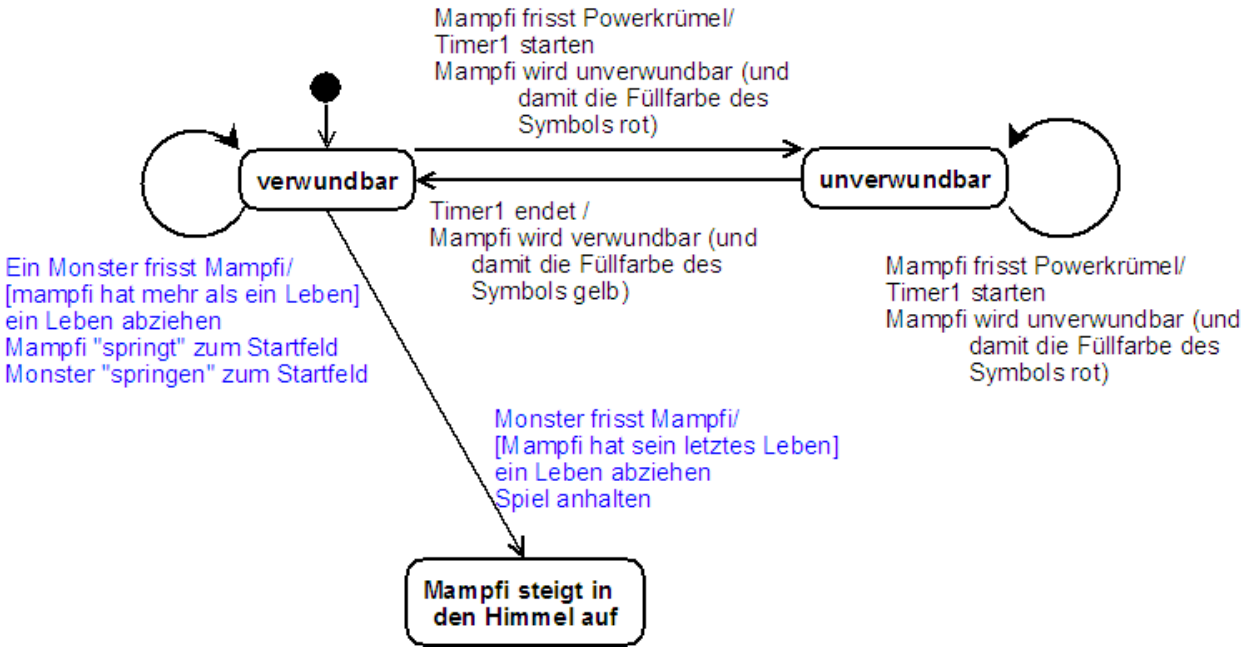


Abbildung 10: Zustandsdiagramm für die Modellierung der Zustände von Mampfi im Kontext des "gefressen werdens".

Aufgabe 19.13
Welche Bedeutung haben die blauen Hervorhebungen bei den Beschriftungen der Zustandsübergäng?
An welchen Stellen muss das BlueJ-Projekt ergänzt werden, um die im Zustandsdiagramm modellierten Abläufe umzusetzen?

Weil die Verwundbarkeit von mampfi die Unverwundbarkeit der Monster bedingt und umgekehrt ist ein großer Teil von dem Zustandsdiagramm aus Abbildung 10 im BlueJ-Projekt bereits umgesetzt. Es fehlen nur noch die Übergänge, die mit blauem Text beschrieben sind.
Änderungen sind nur in der Klasse SPIELSTEUERUNG notwendig und dort hauptsächlich in der Methode SpielzugAuswerten.

Aufgabe 19.14
Setze die noch nicht implementierten Zustandsübergänge aus Abbildung 10 in der Klasse SPIELSTEURUNG um.
Überlege dabei auch, wie man das Spielende erzwingen kann, d.h. dass ein weiterspielen nicht mehr möglich ist, wenn kein Leben vorhanden ist.

Bei Bedarf findest du eine Hilfestellung im Anhang.

Aufgabe 19.15
Das Spielende lässt sich noch deutlicher darstellen. Folgende Möglichkeiten gibt es:
a) Mampfi macht den Mund auf und zu und bei den Monstern bewegen sich die Füße. Das Ende dieser Bewegungen würde zum Spielende gut passen.
b) Mampfi könnte verschwinden, indem der Öffnungswinkel des Kreissektors mit dem Mampfi dargestellt wird Grad für Grad auf 0 reduziert wird und somit das Symbol von Mampfi verschwunden ist.
Versuche diese Ideen umzusetzen oder entwickle eigene.

19.5 Zusammenfassung

Aufgabe 19.16
Fasse die wesentlichen Inhalte dieses Kapitels in deinem Heft zusammen. Folgende wichtigen Begriffe sollten dabei enthalten sein:
logische Operatoren, Zustandsdiagramm insbesondere mit der Erweiterung bei der Beschriftung des Zustandsübergangs.

Anhang A: Tipps zu den Aufgaben

Lösung zu Aufgabe 19.2:
Der folgende kommentierte Quelltextauszug soll verdeutlichen wie man mit Hilfe des Tickers (Taktgeber) des Backends und der Methode *Tick* in der SPIELSTEUERUNG einen Timer implementieren kann.

```
class SPIELSTEUERUNG
{
    // Attribute
     ...
     private int timer1;

     ...
     private STEUERUNGSANZEIGE anzeige;

     //Konstruktor
    public SPIELSTEUERUNG()
    {
          ...
          anzeige = new STEUERUNGSANZEIGE();

     }

    public void AufTasteReagieren(int taste)
    {
          ...
          anzeige.TickerStarten(200);
          ...
     }

     ...
          timer1 = 30;
     ...

    public void Tick()
    {
        if(timer1>0)
        {
            timer1 = timer1 -1;
            if(timer1 == 0)
            {
               // Methodenaufrufe,
               // die am Ende des
               // Zeitintervalls
               // ausgeführt werden sollen
            }

        }
        ...
     }
}
```

Deklaration des Attributs timer1, mit dessen Hilfe ein Zeitintervall festgelegt wird

Über die Steuerungsanzeige kann, wie im letzten Kapitel erläutert, einen Ticker (Taktgeber) gestartet werden.

Starten des Tickers und Festlegen des Taktimpulsintervalls auf 200 ms.

Initialisieren bzw. Zurücksetzen des Timers an geeigneter Stelle: Das Zeitintervall wird hier auf 30 * 200 ms = 6 s festgelegt.

Wenn der Wert des Attributs timer1 gößer als 0 ist, dann wird er bei jedem Tick (Taktimpuls) um eins reduziert.

Wurde 0 als Wert des Attributs timer1 erreicht, ist das Zeitintervall abgelaufen. Sollen am Ende des Zeitintervalls Methoden ausgeführt werden, müssen sie hier notiert werden

Tipps zu Aufgabe 19.10

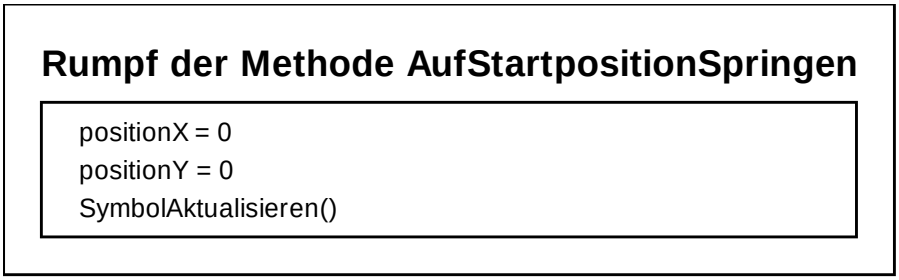
a) Eine gezählten Wiederholung ist nötig.

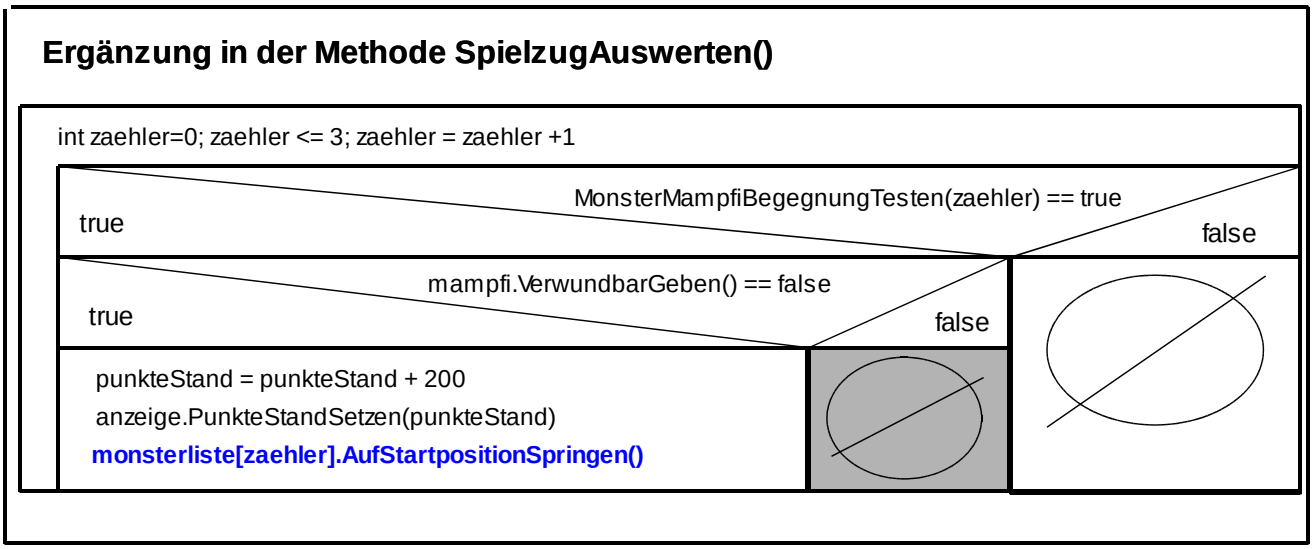
b) Bevor eine Punkteerhöhung stattfindet muss überprüft werden, ob Mampfi unverwundbar ist. Dies ist möglich mit der Bedingung

```
mampfi.VerwundbarGeben() == false
```

Dazu muss in der Klasse SPIELFIGUR die Methode *VerwundbarGeben* implementiert werden.

Tipps zu Aufgabe 19.11





Tipps zu Aufgabe 19.14

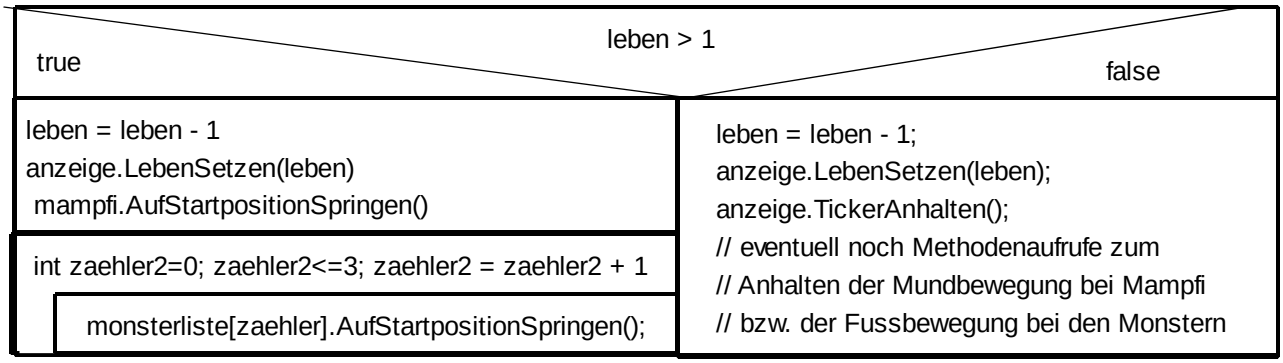
- Der grau gekennzeichnete else Zweig in dem Struktogramm zu Aufgabe 19.10 bzw. 19.11 tritt genau dann ein, wenn ein Monster Mampfi frisst. Er muss nun gefüllt werden.
- Da die ausgelösten Aktionen von einer Bedingung abhängen, muss eine bedingte Anweisung eingefügt werden. Je nachdem ob es das letzte Leben ist oder nicht sind unterschiedliche Methodenaufrufe nötig.
- Die Methode *AufStartpositionSpringen* muss in der Klasse MAMPFI überschrieben (überdeckt) werden, da sonst Mampfi und die Monster auf die Position (0/0) springen. Mampfi muss eine andere Startposition haben. (Wer will, verwendet die im Labyrinth gespeicherte).
- Noch ein Tipp zum Spielende:
  Über `steuerungsanzeige.TickerAnhalten();`
  kann das Spiel angehalten werden. Damit man das Spiel jedoch nicht einfach über die „S“ Taste wieder starten kann, muss in der Methode *AufTasteReagieren* eine bedingte Anweisung ergänzt werden: Ein Start darf nur möglich sein, wenn Mampfi mindestens ein Leben hat.

Noch mehr Tipps zu Aufgabe 19.10
Betrachte den Tipp zu Aufgabe 19.11 ohne die Zeile in blaue Schriftfarbe.

Noch mehr Tipps zu Aufgabe 19.14

In das grau markierte Feld von der Lösung 19.10 bzw. 19.11 muss folgender Programmabschnitt ergänzt werden:

Ergänzung in der Methode SpielzugAuswerten()



Der grau gekennzeichnete else Zweig ist bei der bisherigen Lösung leer und muss dann gefüllt werden, wenn man den Fall betrachtet, dass die Monster Mampfi fressen.