Fluchtwegssimulation von randomisierten Agenten

Franziska Günzinger

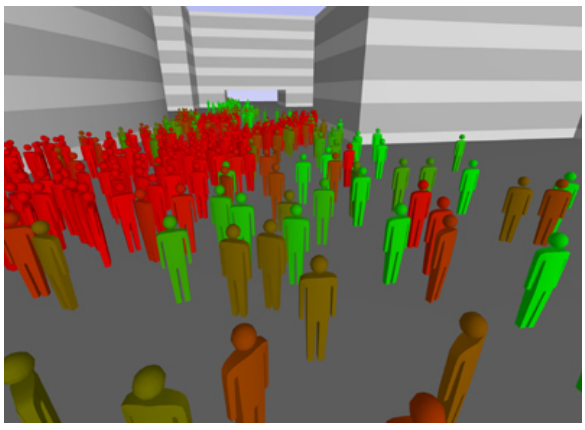

Abbildung 1

Neufahrn 2014

Fluchtwegssimulation von randomisierten Agenten

Franziska Günzinger

W-Seminararbeit zum Rahmenthema
„Zufall und Nichtdeterminismus“
im Leitfach Informatik
am Oskar–Maria–Graf–Gymnasium
Neufahrn b. Freising

vorgelegt von
Franziska Günzinger

betreut durch
StR Baumer

Neufahrn, 11. November 2014

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Das Problem

Das Problem, Menschen aus einem Raum mit nur einem Ausgang zu retten, existiert seitdem es Höhlen, bzw. Räume gibt. Nicht selten kommt es vor, dass sich viele Personen in einem begrenzten Raum aufhalten, z.B. bei Konzerten oder Ausstellungen. Aufgrund unvorhergesehener Ursachen, können Massenpaniken entstehen. In Folge dieser kann es zu Unglucksfällen kommen. Diese gilt es zu verhindern.

Da Massenpaniken unterschiedlich verlaufen, wurden veschiedene Lösungsansätze entwickelt, um solche Situationen zu verhindern:

Einer davon ist, dass Architekten beauftragt werden dieser Situation vorzubeugen, indem für jeden Raum in jedem Gebäude ein eigener Plan für den Fluchtweg kreiert wird. Eine andere Möglichkeit besteht darin, eine Fluchtwegss-imulation anzufertigen. In der hier vorgestellten Simualtion ist von der Größe des Raumes bis hin zur Geschwindigkeit der Agenten alles individuell einstellbar, sodass das Ergebnis möglichst realitätsgetreu ist.

Die Nachteile einer Simulation liegen darin, dass sie zwar viele Bereiche abdecken, aber aufgrund diverser Einflussfaktoren nur ein Scheinbild darstellen können. Es kommt vor, dass der Raum anders aussieht, die Menschen sich anders bewegen und sich unerwartet Objekte im Raum befinden, die nicht mit einkalkuliert werden können.

Folgende Situation herrscht in dem für die Seminararbeit entwickeltem Programm:

In einem Raum befindet sich eine feste Anzahl an Menschen und es gibt einen, bzw. zwei Notausgänge. An der Westwand ensteht ein Feuer, welches sich nach und nach ausbreitet. Nun kommt es zum Stromausfall, was zur absoluten Dunkelheit im Raum führt.Die Menschen sehen nichts außer dem Feuer und der Notbeleuchtung des Notausganges. In ihrer aufkeimenden Panik laufen sie wahllos durch den Raum und weichen lediglich dem Feuer aus. Im variierbaren Radius um die Notausgangsbeleuchtung finden sie den Ausgang.

Aufgrund dieser Situation stellt sich die Frage:

Wie lange braucht man um alle bzw. möglichst viele Agenten zu retten?

2 Fluchtwegsmiulation von randomisierten Agenten

2.1 Grundlagen

Bevor das Programm geschrieben werden konnte, wurden ein paar Begriffe definiert:

Fluchtweg: [BfG14]

> Fluchtwege sind Verkehrswege, an die besondere Anforderungen zu stellen sind. Sie dienen der Flucht aus einem möglichen Gefährdungsbereich und in der Regel zugleich der Rettung von Personen. Fluchtwege führen ins Freie oder in einen gesicherten Bereich. Fluchtwege sind auch die im Bauordnungsrecht definierten Rettungswege, sofern sie selbstständig begangen werden können.

Simulation: [Ber93]

> [lat.], [...] 3. Technik: allg. eine modellhafte Nachbildung eines Systems mit Hilfe eines Ersatzsystems, wobei die mit dem simulierten System gewonnenen Ergebnisse möglichst mit denen des ürsprüngl. Systems übereinstimmen sollen. [...]

randomisiert: [Joa14]

> Randomisierung
>
> wahllos herausgreifen (für Experimente) [engl. randon, random "zufällig", at random "aufs Geratewohl, auf gut Glück, wahllos", eigtl. "in großer Eile"]

Agent: [SF14]

> 1. Informatik: auch Software-Agent, Programm, das als Bestandteil eines verteilten Systems selbstständig handelt und mit anderen Agenten des Systems kommuniziert. Eine Software wird als Agent bezeichnet, wenn sie die folgenden fünf Eigenschaften besitzt: (a) autonomes Handeln, d.h. der Agent handelt ohne oder nur mit sehr geringem Benutzereingriff; (b) Proaktivität, d.h. der Agent führt initiativ eigene Aktionen aus; (c) Reaktivität, d.h. der Agent reagiert selbstständig auf Änderungen der Umwelt; (d) soziales Handeln, d.h. der Agent kann mit anderen Agenten kommunizieren; (e) Lernfähigkeit, d.h. der Agent baut im Laufe der Zeit ein eigenes Wissen auf, das er für spätere Entscheidungen heranzieht. [...]

Anschließend wurde das Programm in zwei Phasen unterteilt:

Phase I	Monat	Zeitaufwand
a) Ab welchem Radius sieht man die Tür?	April	2 Stunden
b) Bis zu welchem Abstand (in Metern) weicht man dem Feuer aus?	April	2 Stunden
c) Wie groß ist der Raum?	April	3 Stunden
Phase II		
a) Wie schnell breitet sich das Feuer aus?	Mai	4 Stunden
b) Wieviele Fluchtwege gibt es?	August	2 Stunden

Abbildung 2

Die einzelnen Schritte werden in 2.2.2 genauer erklärt.

In den folgenden Kapiteln wird genauer auf das Programm, die Einschränkungen, Probleme und Lücken im Programm, sowie die dazu entnommenen Statistiken eingegangen.

2.2 Implementierung

Es wurden eigens für die spezifischen Anwendungen diverse Variablen angelegt, um den Benutzern die passenden Variationen zu ermöglichen.

Die Simulation spielt sich in einem quadratischem Raum ab. Je nach Wunsch kann man einstellen, ob ein Ausgang, bzw. zwei Ausgänge existieren. Das Feuer breitet sich immer von West nach Ost aus, dabei ist die Geschwindigkeit mit der es sich ausbreitet einstellbar. Auch die Anzahl der Agenten kann variiert werden, um eine möglichst reale Situation darzustellen. Auch ist es möglich eine Mauer einzuziehen. Diese kann man allerdings nur an der nördlichen, östlichen und südlichen Wand befestigen, da es keinen Sinn ergibt, wenn sie sich von Anfang an im Feuer befindet. Die Mauer erschwert es den Agenten, zusätzlich zum Stromausfall und dem Feuer, den Ausgang zu finden.

Insgesamt existieren zwei verschiedene Variationen, die sich in der Bewegung der Agenten unterscheiden. Sie wurden entwickelt, da sich Menschen in Notsituationen unterschiedlich bewegen können, jedoch dann meist aber einem Muster folgen.

In der ersten Variation bewegen sich die Agenten wahllos in alle vier Himmelsrichtungen, da sie nichts sehen. Gelangen sie ans Feuer gehen die Agenten immer in die entgegengesetzte Richtung von diesem, sprich nach Osten. Identisch verhalten sie sich, wenn die Agenten an eine Wand stoßen. Je nachdem an welche sie gelangen, bewegen sie sich nach Süden, Westen oder Norden.

In der zweiten Variation verhalten sich die Agenten gleich, allerdings tasten sie sich, sobald sie an eine Wand stoßen, an dieser weiter. Dies machen sie solange bis sie den Ausgang erreichen.

Allgemein wird der Raum wie folgt dargestellt:

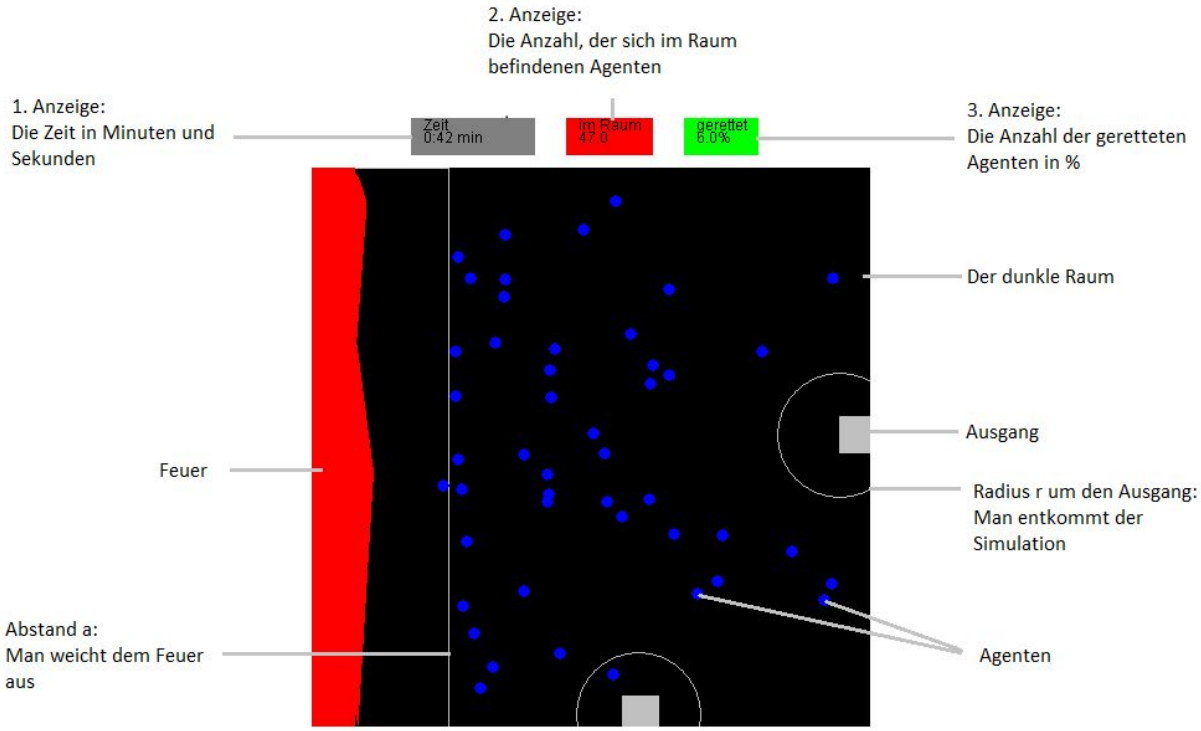

Abbildung 3

2.2.1 Variablen

Der Code enthält viele verschiedene Variablen, manche sind allerdings nur dazu da diverse Sachen zu vereinfachen, deswegen wird nicht näher auf diese eingegangen. Die wichtigsten fünf Variablen sind *int anzahl*; *int größe*; *int curFeuerX*; *int radius* und *double gerettet*.

1. *int anzahl*

   Bestimmt die Anzahl an Personen, die sich in dem Raum befinden sollen. Zu dieser gehört die Variable *Agent Agentenarray []*. Hier werden die einzelnen Agenten gespeichert, allerdings nur so viele, wie es die *anzahl* vorgibt.

2. *int größe*

   Bestimmt die Größe des Raumes und gehört auch zum Typ int. Da die Variable für das zweidimensionale Array (*int [] [] Raum*) dazugehört, ist der Raum immer quadratisch.

3. *int curFeuerX*

   Legt die Verbreitungsrate des Feuers fest. Es wurde eingestellt, dass das Feuer sich jede x-te Runde ausbreitet. Somit breitet es sich schneller aus, wenn weniger Agenten

im Raum sind. Da sich Feuer in der Realität nicht gleichmäßig ausbreitet, wurde versucht die Simulation der Realität anzupassen.

4. *int radius*

   Gibt die Hälfte der Größe eines einzelnen Agenten an. Radius bedeutet in diesem Fall, dass die Agenten als Kreise dargestellt werden. Auch hier gehört die Variable zum Typ int.

5. *double gerettet*

   Gibt die Anzahl aller geretteten Agenten aus. Dies ist die einzige wichtige Variable die dem Typ double angehört. In Abbildung existieren drei Anzeigen. In der letzten werden die geretteten Agenten als Prozentzahl ausgegeben, daher muss hier eine Kommazahl vorliegen.

Neben diesen Variablen gibt es noch weitere, die zum einem zum Erstellen der Mauer notwendig sind, zum anderem zum Hochzählen der Zeit und der Runden.

Die vier Variablen *int xMauer*, *int yMauer*, *int bMauer* und *int lMauer* legen die einzelnen Werte der Mauer fest. *xMauer* legt die x-Koordinate fest, an dem sich die obere linke Ecke der Mauer befindet. *yMauer* verhält sich wie die Variable *xMauer*, allerdings legt sie die y-Koordinae fest. *bMauer* bestimmt die Breite, sowie *lMauer* die Länge der Mauer. Alle vier Variablen sind im Konstruktor veränderbar.

Die anderen vier Variablen *int countSec*, *int sec*, *int min* und *int count* sind Hilfsvariablen. Die ersten drei zählen die Zeit mit. In 2.2.3 wird unter Code genauer erklärt, inwieweit diese drei Variablen wichtig sind. *int count* hingegen zählt die Runden mit. Am Ende der Methode *run()* wird die Variable hochgezählt. Anschließend wird überprüft, ob die Variable einen bestimmten Wert erreicht. Ist dies der Fall bewegt sich das Feuer weiter. Die Variable *curFeuerX* wird um einen bestimmten Wert, den man individuell einstellen kann, erhöht. *int count* hingegen wird auf den Wert eins zurückgesetzt.

2.2.2 Die Phasen

Der Code des Programms basiert auf dem für die Seminararbeit aufgestellten Phasenplan. Dieser wurde in Kapitel 2.1 bereits dargelegt. In Phase I wurden die wichtigsten drei Grundeinstellungen festgelegt.

1. Radius

   Der Radius ...

   a) ... soll individuell einstellbar sein.

   b) ... sollte mindestens 2 m betragen.

   c) ... sollte maximal 10 m betragen.

   Dabei wurde festgelegt, dass ein Meter etwa 20 Pixel entspricht. Nachdem diese drei

Schritte beachtet wurden kam folgendes Ergebnis zustande:

Der Radius ist variierbar und in der Grafik als Kreis dargestellt. Da dies in der Umsetzung sehr aufwendig gewesen wäre, wurde der Kreis zu einem Rechteck vereinfacht, welches sich im Kreis der Grafik befindet. Befindet sich der Agent in diesem Ausgangsbereich, verlässt er automatisch die Simulation.

2. Abstand zum Feuer

   Der Abstand zum Feuer ...

   a) ... soll individuell einstellbar sein.

   b) ... soll sich linear mit der Ausbreitung des Feuers bewegen.

   c) ... sollte mindestens 1 m betragen.

   d) ... sollte maximal 5 m betragen.

   An dieser Stelle des Codes *g.drawRect(50, 50, curFeuerX+50, 450);* wird die Entfernung des Abstandes zum Feuer festgelegt. Die +50 bei curFeuerX+50 sagt aus, dass der Abstand 50 Pixel vom Feuer entfernt sein soll, sprich 2,5 m. Befindet sich ein Agent innerhalb der Abstandszone, geht er automatisch nach Osten, um dem Feuer zu entkommen.

3. Raum

   Als letzter Punkt in Phase I wurde die Größe des Raumes festgelegt. Zwar ist auch sie einstellbar, ist das Programm jedoch aktuell so geschrieben, dass die Variablen auf eine Raumgröße von $größe = 500$ abgestimmt sind.

   In Phase II wurde die Geschwindigkeit mit der sich das Feuer ausbreitet und die Anzahl der Fluchtwege festgesetzt. Wie in 2.2.1 schon beschrieben breitet sich das Feuer jede x-te Runde aus. Auch kann man über die Variabel *curFeuerX* die Breite festlegen, z.B. ob sich das Feuer um 20 Pixel vergrößert. In diesem Code wurden maximal zwei Ausgänge festgelegt, allerdings ist es möglich noch mehr Ausgänge zu programmieren.

2.2.3 Der Code

Das Programm besteht aus zwei Klassen. Die kleinere der beiden ist die Klasse *Agent*. Die erste Klasse bestimmt die Anfangskoordinaten der Agenten. Dazu wird die Formel *(int) (((Math.random())*375+100)* benutzt. Der Zahlenbereich geht von 100 bis 475. Außerdem wird auch in dieser Klasse mit Hilfe der Variable *boolean foundWayOut* festgelegt, ob der Agent der Simulation bereits entkommen ist oder nicht. Ist er bereits entkommen, wird die Variable auf "true"gesetzt und der Agent wird nicht weiter beachtet. Um einen Fehler zu vermeiden, der in 2.3.2 genauer erklärt wird, wird jedem Agenten auch eine zufällige Richtung durch die Variable

*int richtung = (int) (((Math.random())*4+1)* zugeteilt. Die Klasse Agent verfügt über drei Methoden. Alle drei sind Getter. Sie geben die x-, sowie die y-Koordinate und die Richtung aus.

Die andere Klasse ist die Main-Klasse. Von ihr existieren jedoch, wie bereits in 2.2 erklärt, zwei Versionen:

Fluchtwegsimulation I und Fluchtwegsimulation II

Beide Klassen sind in etwa gleich aufgebaut, da sie sich ausschließlich in der Bewegung der Agenten unterscheiden. Das bedeutet, dass die Methode *run()* sich im Code unterscheidet. Neben dieser Methode gibt es weitere fünf sowie den Konstruktor (die Main-Methode wurde nicht mitgezählt).

Im *Konstruktor* werden alle globalen Variabeln definiert und die Agenten werden anhand einer for-Schleife im Array gespeichert. Auch die Mauer wird hier erstellt.

Die erste Methode ist *paint()*. In dieser Methode wird alles wichtige in die Grafik eingefügt. Der Raum wird erstellt, das Feuer, die Ausgänge, die Zeitanzeige, die Anzeige wiviele Agenten sich noch im Raum befinden, die Anzeige, wieviele Personen schon gerettet wurden und alles was sonst noch zu sehen ist. Diese Methode ruft auch die nächste auf.

In der Methode *Fuellen()* wird der Agent erzeugt. Wie in 2.2.1 bereits erwähnt wird die Anzahl der Agenten durch die Variable *int anzahl* festgelegt. Anhand einer weiteren for-Schleife wird überprüft, ob sich der Agent noch in der Simulation befindet. Ist dies der Fall, wird er auf die ihm zugewiesenen Koordinaten platziert, sprich, in der Grafik erscheint er nun als blauer Kreis.

Die nächste Methode ist ein einfacher Dreizeiler. Die Methode *neuberechnen()* weist dem Agenten eine neue Richtung zu. Sie wird immer wieder in der Methode *run()* aufgerufen, da der Agent sich nicht immer frei entscheiden kann in welche Richtung er als nächstes gehen will.

Die Methode *start()* hat nur einen Zweck. Sie startet die wichtigste Methode der Klasse, die Methode *run()*.

Im Code der Methode *run()* wird die x-Koordinate als a und die y-Koordinate als b bezeichnet. Am Ende jedes Schrittes (hier: wenn der Agent seinen Zug beendet hat), werden die neue Richtung, sowie die Koordinaten im Array gespeichert, damit derselbe Agent auch den ihm zugewiesenen Schritt durchführt. Der Code ist in eine for-Schleife gebettet, die alle Agenten einzeln durcharbeitet. In dieser werden zuerst die Variablen festgesetzt. Auch läuft hier die Zeit mit, dass heißt, dass nach jedem Schritt eines Agenten 50 Millisekunden vergehen. Jeder Schritt wird von einer weiteren Variable (*int countSec* mitgezählt. In einem if wird überprüft, ob die Variable *countSec* den Wert 20 erreicht hat (da 20 mal 50 Millisekunden einer Sekunde

gleichkommen). Ist dies der Fall wird die Variable *int sec* um eins hochgezählt. In einem weiterem if wird überprüft, ob die Variable *sec* den Wert 60 erreicht hat, dann wird die Variable *int min* um eins hochgezählt. Nach jedem if, wird die Variable, die überprüft wird, wieder auf null gesetzt. Die for-Schleife wiederum befindet sich in einer while-Schleife. Die while-Schleife endet, wenn das Feuer den ganzen Raum ausfüllt. Die Methode bricht ab. Da der Agent in vier Richtungen gehen kann gleicht jeder der vier Richtungsabschnitte dem anderem.

run() in Fluchtwgsimulation I:

Am Anfang jedes Richtungsabschnitts wird überprüft, ob die Richtung dem jeweiligem Abschnitt entspricht. Die Ziffer eins für den Norden, zwei für Osten, drei für Süden und die Ziffer vier für den Westen.

Abbildung 4

Anschließend wird in einem if überprüft, ob sich der Agent noch in der Simulation befindet oder nicht. Hat der Agent bereits den Ausgang erreicht und damit die Simulation verlassen, wird er nicht weiter beachtet. Bewegt sich der Agent nach Norden, so bleibt die x-Koordinate gleich und die y-Koordinate verringert sich um den Radius. Jeder Schritt des Agenten wird in dieser Methode überprüft. Befindet sich die neue Zelle, die er betreten soll, außerhalb des Raumes, so bewegt er sich in die entgegengesetzte Richtung. Es muss überprüft werden, ob das Feld auf das der Agent gehen will bereits durch einen anderen blockiert wird oder noch frei ist. Befindet sich ein Agent darauf, so wird das Feld mit der Ziffer eins beschriftet. Das Feld gilt als besetzt und der Agent bekommt eine neue Richtung zugewiesen. Als letztes wird noch überprüft, ob die erstellte Mauer den nächsten Schritt verhindert. Auch hier bekommt der Agent eine neue Richtung zugewiesen. Ist die Zelle frei und innerhalb des Raumes, führt der Agent seinen Schritt aus. Die neue Koordinate wird auf eins gesetzt, die jetzige bekommt die Ziffer null. Die Koordinate die verändert wird, wird neu gesetzt. Im Code sieht die Zeile folgendermaßen aus: *b = b - radius;*. Im Anschluss wird die Methode *repaint()* aufgerufen und in der Methode *try()* wird bestimmt, dass nach dem Schritt 50 Millisekunden gewartet werden soll. Kann die Methode nicht ausgeführt werden, wird sie mit Hilfe der Methode *catch(InterruptedExcepion e)* aufgefangen. Bevor dem Agenten eine neue Richtung

zugewiesen werden kann, müssen weitere Angaben, bzw. Vorgaben überprüft werden. Befindet sich der Agent nun hinterhalb der Sichtline vom Feuer, so bewegt er sich im nächsten Schritt nach Osten, also vom Feuer weg. Befindet sich der Agent innerhalb der Zone um den Ausgang, so verlässt er die Simulation durch den Notausgang und die Zahl der Geretteten erhöht sich um eins. Ansonsten bekommt der Agent eine zufällige Richtung zugewiesen. Am Ende jedes Richtungsabschnitts wird die Variable *countSec* hochgezählt.

run() in Fluchtwegsimulation II:

Im Allgemeinen ähnelt die Methode der vorhergehenden *run()* Methode, allerdings muss hier mehr beachtet werden, da der Agent an der Wand weitergehen soll. Am Anfang wird überprüft, ob der Agent sich noch in der Simulation befindet, oder ob er bereits den Ausgang erreicht und damit die Simulation verlassen hat. Anschließend wird überprüft, ob sich der Agent im Bereich der Außenmauer aufhält. Entspricht seine alte Richtung der aktuellen, wird überprüft, an welchem Ort sich die neue Koordinate innerhalb des Raumes befindet. Wenn sich der Agent z.B. innerhalb des Bereiches der Nordmauer aufhält, bekommt er entweder Richtung zwei oder vier zugewiesen. Anschließend wird für beide Richtungen Schema duchgeführt:

Zuerst wird ermittelt, ob sich bereits ein anderer Agent auf der neuen Koordinate befindet. Ist das Feld besetzt, geht der Agent in die entgegengesetzte Richtung. Ansonsten wird überprüft, ob sich der Agent in einer Ecke befindet. Ist dies der Fall, wird folgendes Verfahren angewendet:

```
} else if (a >= 465 && b <= 60) {
    if (Raum[a][b+radius] == 1) {
        richtung = 4;
    } else {
        Raum[a][b+radius] = 1;
        Raum[a][b] = 2;
        b = b+radius;
        repaint();
        try {
        Thread.sleep(50);
        } catch (InterruptedException e) {}
        richtung = 3;
    }
```

Abbildung 5

Zuletzt wird ermittelt, ob sich der Agent bereits im Ausgang befindet. Dann verlässt der Agent die Simulation und die Zahl der Geretteten erhöht sich um eins.

Tritt alles nicht ein, bewegt sich der Agent auf das nächste Feld. Hierbei wird wie bei der Methode *run()* Fluchtwegsimulation I verfahren. Die Richtung wird nach dem Schritt nicht verändert, da es das Ziel des Agenten ist, an der Mauer weiter zu laufen. Ist die y-Koordniate größer als 50 wird nach dem gängigen Prinzip verfahren. Ist der Platz belegt, bekommt der Agent eine neue Richtung zugewiesen, ansonsten bewegt

er sich auf das Feld. Entspricht die neue Richtung nicht der alten, muss überprüft werden, ob der Agent bereits davor an der Wand entlang gegangen ist. Je nachdem aus welcher Richtung er kam, geht er nun in dieser weiter. Befindet sich der Agent allerdings mitten im Raum, wird auch hier das Prinzip aus *Fluchtwegsimulation I - run()* angewendet. Am Ende jeder Richtung wird der Variable *richtungalt* die aktuelle Richtung zugewiesen und die Variable *countSec* wird um eins hochgezählt.

2.3 Einschränkungen, Probleme und Lücken im Programm

Im Folgenden werden die Einschränkungen, Probleme und Lücken, die im Programm vorhanden sind aufgeführt und beschrieben.

2.3.1 Einschränkungen

Wie bereits in 2.2.2 beschrieben, tritt folgende Einschränkung auf: Der Radius r um den Ausgang wird auf der Grafik als Kreis, bzw. Halbkreis, dargstellt, im Code hingegen, wird mit einem sich im Kreis befindlichem Rechteck gerechnet. Daraus folgt, dass sich der Agent in der Grafik zwar schon innerhalb des Radius befindet, die Simulation aber noch nicht verlassen kann, da er sich noch nicht innerhalb des Rechtecks bewegt.

Eine weitere Einschränkung ist, dass es auf der Grafik so aussieht, als würden die Agenten aufeinander stehen. Dies ist aber nicht der Fall. In Wirklichkeit geht der Code davon aus, dass der Agent genau ein Pixel groß ist, der restliche Kreis ist nur Zierde. Deswegen überlappen sich die Agenten teilweise können sich aber nicht eins zu eins auf den anderen stellen.

2.3.2 Probleme

Ein schwerwiegendes Problem ergibt sich dadurch, dass die Rechnerleistung der Rechner, bei denen die Simulation bisher getestet wurde, zu gering ist. Je mehr Agenten sich in einem Raum befinden, bzw. je mehr Dinge berücksichtigt werden müssen, desto mehr Leistung ist erforderlich. Deswegen schafft es die Simulation teilweise nicht, dass das Feuer die Ostwand erreicht, sie bricht ab.

Ein anderes Problem, dass im Laufe des Programmierens gelöst wurde, war die Zeitangabe. Bei dem Versuch sie extern mitlaufen zu lassen, bewegten sich die Agenten maximal alle zwei Minuten einmal. Ein weiterer Versuch bestand darin, dass die Agenten selbst die Zeit mitzählen. Hierbei wurde aber nicht beachtet, dass nachdem sich ein Agent bewegt hat, eine Sleeptime (im Code: *Thread.sleep(50)*) eingebaut wurde. Die Sleeptime bedeutet, dass der nächste Agent mit seinem Schritt 50

Millisekunden wartet. Unter Berücksichtigung dieser sieht der Code nun wie folgt aus:

```
if (countSec == 20) {
    sec++;
    countSec = 0;
}
if (sec == 60) {
    min++;
    sec = 0;
}
```

Abbildung 6

2.3.3 Lücken

Wie schon mehrmals erwähnt, ist der Raum ein Quadrat und kann sich deswegen nicht an runde oder vieleckige Räume anpassen. Hätte man dies programmiert, hätte es den Rahmen einer Seminararbeit gesprengt.

Des weiteren wurde nicht berücksichtigt, dass Objekte im Raum liegen, bzw. von der Decke fallen können. Es wurde nur insofern auf diesen Punkt eingegangen, dass eine Mauer integriert werden kann.

Das Programm entspricht auch in dem Sinne nicht der Realität, dass alle Agenten gleich groß sind. Es gibt dünne und dicke Menschen, im Programm wird allerdings ein Durchschnitt genommen. Außerdem bewgen sich Menschen auch unterschiedlich schnell, manche spazieren, wieder andere laufen. In der Simulation bewegen sich die Agenten jedoch alle in derselben Geschwindigkeit.

Dirk Helbing, Illés Farkas und Tamás Vicsek schreiben in ihrer Arbeit "*Simulating dynamical features of escape panic, Nature Vol. 407*", die am 28. September 2000 veröffentlicht wurde, hauptsächlich über die Simulation von flüchtenden, in Panik geratenen Menschen [HFV00].

In der Grafik der Simulation ist um den Ausgang ein Kreis zu sehen. Geraten die Agenten in diesen entkommen sie der Simulation. Wie jedoch in Kapitel 2.2.2 beschrieben, wird im Code mit einem Rechteck, anstatt eines Kreises gerechnet. Zur Bewältigung des Kreisproblemes kann man den Satz des Pythagoras verwenden:

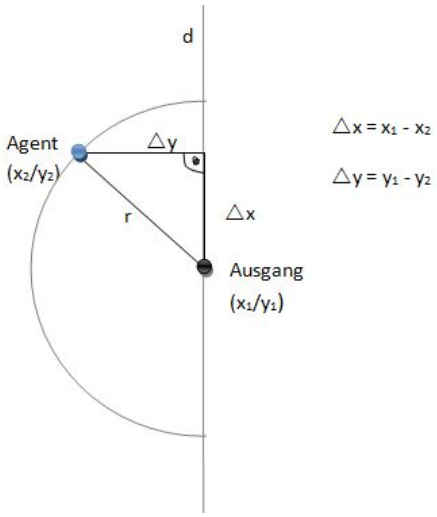

Abbildung 7

Der Satz des Pythagoras wird benutzt, um den Abstand zwischen dem Agenten und dem Ausgang zu berechnen, hier r. Allgemein kann man den Pythagoras nur verwenden, wenn es sich um ein rechtwinkliges Dreieck handelt. Dieses kann man erzeugen, wenn man z.B. einen Halbkreis vorliegen hat. Auf der Senkrechten (d), die den Kreis halbiert, wird Δ x aufgetragen. Durch den Punkt des Agenten wird auf die Senkrechte ein Lot errichtet. Der Abstand zwischen dem Agenten und der Senkrechten wird als Δ y bezeichnet.

Um herauszufinden, ob sich der Agent im Bereich um den Ausgang befindet wird nun folgende Formel verwendet:

$(x_1 - x_2)^2 + (y_1 - y_2)^2 <= r^2$

Da es sich als schwierig erweist aus r^2 die Wurzel zu ziehen, bleibt dieses als solches stehen.

3 Statistiken

Um möglichst genaue Durchschnittswerte zu bekommen, wurde die Simulation einen Monat lang jeden Tag jeweils fünf mal mit 10, 25 und 50 Agenten in beiden Simulationen durchgeführt. Dabei fiel auf, dass sich die Durchschnittswerte bereits am Anfang herausbildeten.

Die Mauer wurde nicht offiziell in die Statistik miteingebaut, da die Leistung der zur Verfügung gestandenen Rechner zu gering ist. Allerdings kann man auch ohne diese rechnersich vorhersagen inwieweit die Mauer das Entkommen der Agenten verhindert. Verringert man die Größe des entstehenden Raumes zwischen der eingefügten Mauer und der Wand, so ist die Wahrscheinlichkeit, dass ein Agent in diesen Abschnitt gerät prozentuell geringer, als wenn der entstehende Raum größer wäre. Identisch verhält es sich mit der Länge der Mauer. Allerdings entkommt der Agent

schneller, wenn es sich um einen kleinen Raum handelt.

3.1 Messwerte und Ergebnisse

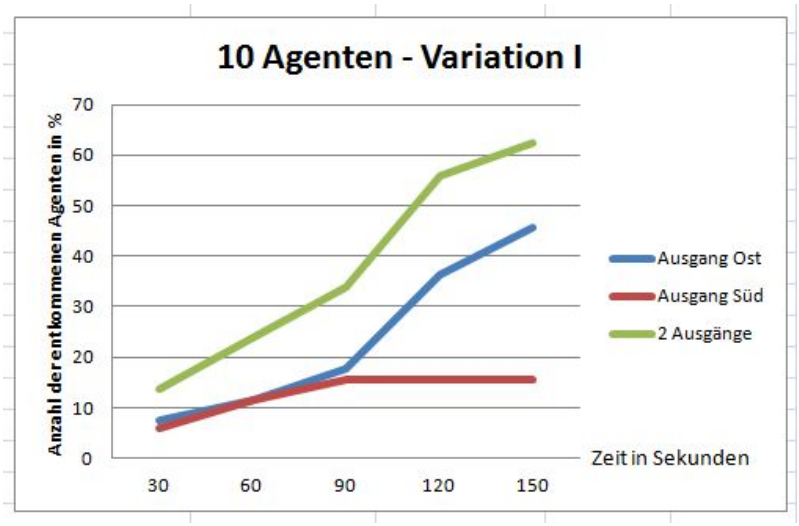

Abbildung 8

Der Graph der Funktion *Ausgang Ost* nimmt erst schwach zu. Nach 30 Sekunden sind im Schnitt weniger als zehn Prozent aller Agenten entkommen. Etwa nach der Hälfte der Zeit (90 Sekunden) steigt die Kurve rasant an und gegen Ende der Simulation flacht diese wieder ab. In diesen 60 Sekunden entkommen in der ersten Hälfte etwa 20% und in der zweiten Hälfte nur noch etwa 10% der Agenten. Ähnlich verhält es sich mit der Funktion *2 Ausgänge*. In den ersten 30 Sekunden schaffen es mehr als 10% der Agenten den Ausgang zu finden und nach 90 Sekunden sind bereits knapp über 30% entkommen. Nach 120 Sekunden schaffen es ungefähr 57%. Die Funktion *Ausgang Süd* nimmt in den ersten 90 Sekunden mit einem stetigem Anstieg von etwa 6% zu, dann stagniert die Kurve. Am Ende der Simulation schaffen es bei zwei Ausgängen über 60%, bei dem Ost-Ausgang knapp 45% und bei dem Süd-Ausgang 18% aller Agenten zu entkommen.

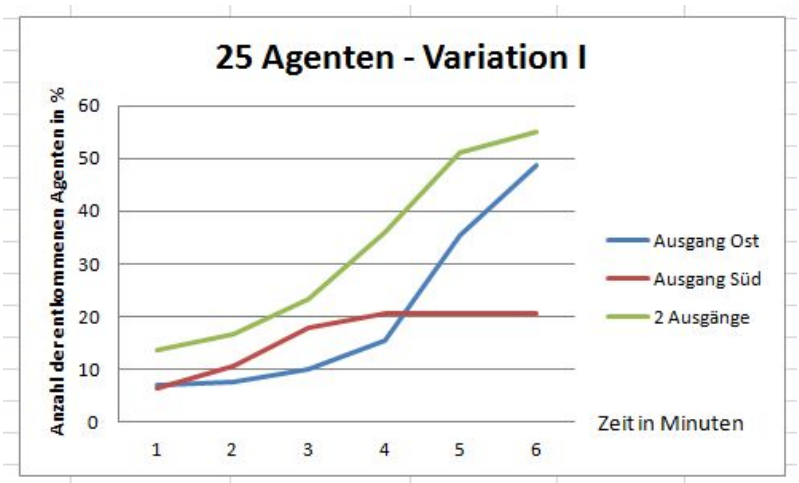

Abbildung 9

Die Graphen der drei Funktionen verhalten sich fast identisch wie in Abbildung 8. Auch hier nimmt sowohl die Funktion *2 Ausgänge*, als auch die Funktion *Ausgang Ost* in den ersten vier bis fünf Minuten exponentiell zu, danach flachen die Kurven ab. Die Funktion *Ausgang Süd* stagniert wiederum etwa nach der Hälfte der Zeit (drei bis vier Minuten), nach einem geradezu fast linearem Anstieg. Etwa 55% der Agenten entkommen am Ende der Simulation, wenn zwei Ausgänge vorhanden sind. Bei dem Ost-Ausgang entkommen knappe 50% und beim Süd-Ausgang schaffen es nur 20% zu entkommen.

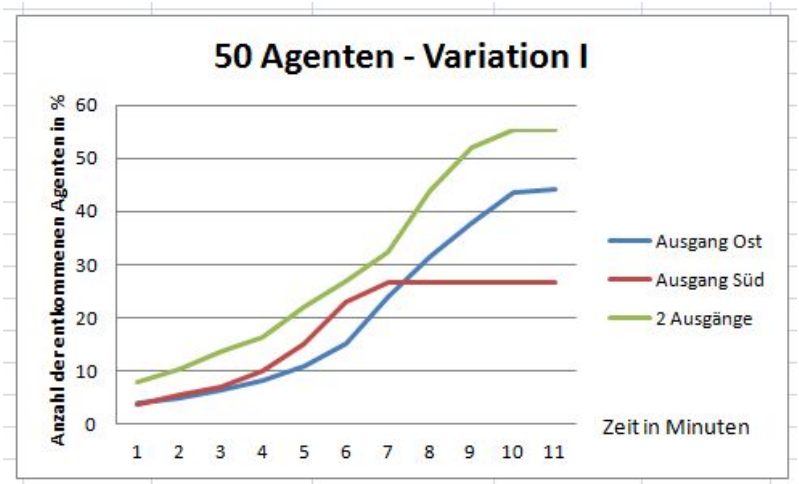

Abbildung 10

Spätestens bei diesem Diagramm wird deutlich, dass alle Fuktionen bis zu verschiedenen Punkten exponentiell ansteigen und dann teilweise aprubt stagnieren, bzw. relativ schnell abflachen. Auch hier sind die Endwerte ähnlich der Simulationen mit weniger Agenten. Bei zwei Ausgängen entkommen etwa 55%, beim Ost-Ausgang etwa 45% und beim Süd-Ausgang etwa 27%.

Dass sich die Prozentzahlen der entkommenen Agenten beim Süd-Ausgang unterscheiden, liegt daran, dass mehr Agenten eine höhere Wahrscheinlichkeit haben zu diesem Ausgang zu gelangen.

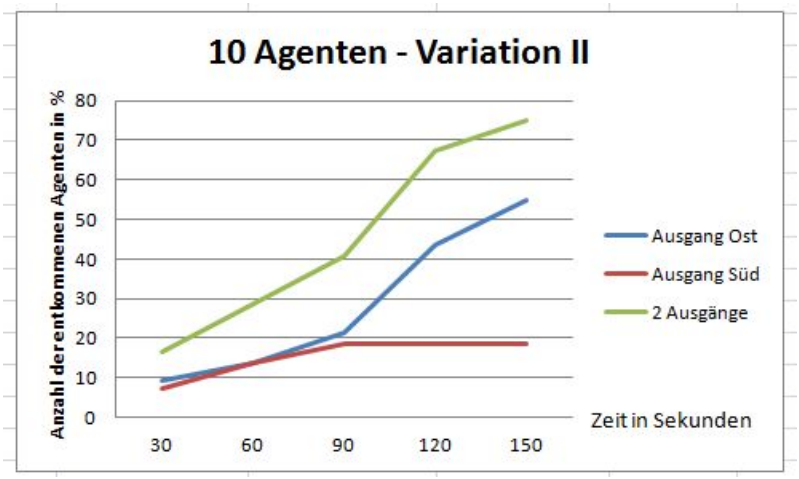

Abbildung 11

Bei der Funktion *2 Ausgänge* sind nach 30 Sekunden über 15% der zehn Agenten entkommen. In den nächsten 60 Sekunden nimmt die Zahl um 10% je 30 Sekunden zu. Danach gibt es einen Anstieg um ca. 30%. Bis zum Ende der Simulation konnten 75% der Agenten aus dem Raum gerettet werden. Bei der Funktion *Ausgang Ost* entkamen nach eineinhalb Minuten alle 30 Sekunden etwa 7% der Agenten. Nach 120 Sekunden wurden bereits 45% gerettet und bis zum Ende schafften es noch weitere 10% dem Feuer zu entkommen. Linear steigt auch die Anzahl der entkommenen Agenten der Funktion *Ausgang Süd* bis einschließlich 90 Sekunden. Danach stagniert die Zahl bei etwas unter 20%.

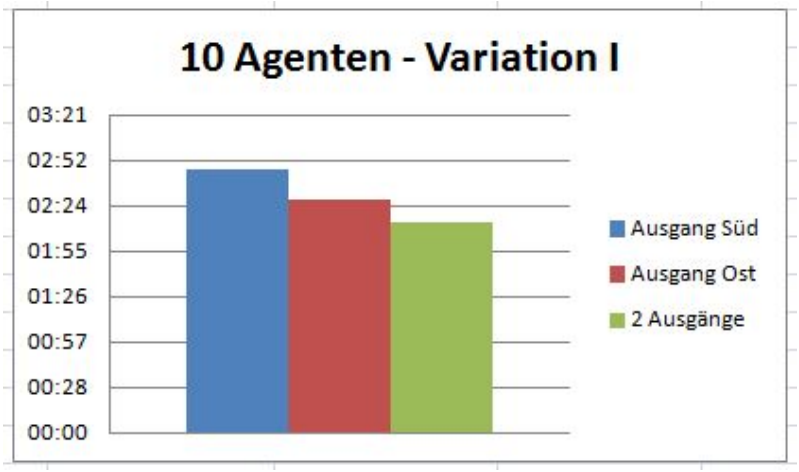

Abbildung 12

Exisitiert nur ein Ausgang an der Süd-Wand brauchen das Feuer ungefähr zwei Miuten und 48 Sekunden um den Raum auszufüllen. Bei einem Ausgang gegenüber des Feuers ist die Simulation nach ca. zwei Minuten und 25 Sekunden beendet. Beinhaltet der Raum zwei Ausgänge hat das Feuer bereits nach zwei Minuten und 10 Sekunden den Raum in Besitz genommen. Da sich das Feuer wie in Kapitel 2.2.1 beschrieben nicht gleichmäßig ausbreitet, nimmt die Zeit, die die Simulation benötigt, mit der Anzahl der entkommenen Agenten exponentiell ab.

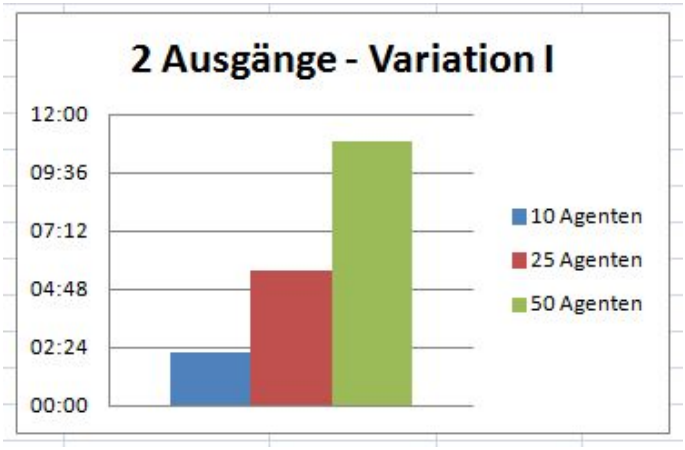

Abbildung 13

Lässt man die Simulation mit zehn Agenten durchlaufen, benötigt das Feuer im Schnitt etwa zwei Minuten und 23 Sekunden um den Raum auszufüllen. Bei 25 Agenten beträgt die Zeit bereits knapp fünf Minuten und bei 50 Agenten braucht das Feuer im Schnitt zehneinhalb Minuten. Betrachten man die Abbildung, fällt auf, dass die Zeit exponentiell mit der Anzahl der Agenten zunimmt.

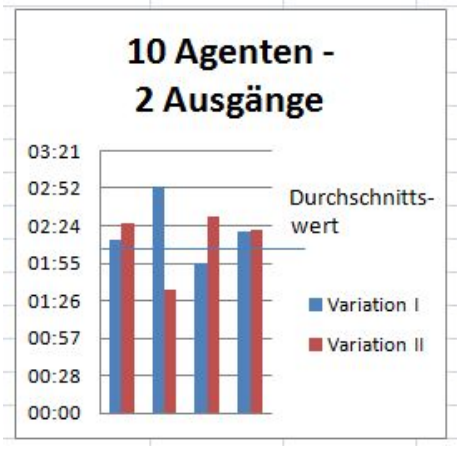

Abbildung 14

Bei der Gegenüberstellung der beiden Varianten konnte beobachtet werden, dass sich die durchschnittlichen Durchlaufzeiten annähernd gleichen.

3.2 Auswertung der Statistik

Folgende Schlüsse kann man daraus ziehen:

a) Rolle des Ausgangsortes:

J nachdem an welchem Ort sich der Ausgang befindet, haben die Agenten eine andere Chance diesen zu erreichen. Beim Süd-Ausgang stagniert die Anzahl der entkommenen Agenten nach einer gewissen Zeit, da das Feuer den Ausgang versperrt. Dementsprechend entkommen auch insgesamt weniger Agenten, als wenn der Ausgang sich auf der gegenüberliegenden Seite des Feuers befindet.

b) Mauer:

Je größer der freigelassene Raum zwischen Mauer und Wand ist, desto größer wird die Wahrscheinlichkeit, dass sich der Agent dorthin verirrt und nicht wieder herausfindet. Je kleiner der freigelassene Raum zwischen Mauer und Wand ist, desto kleiner wird die Wahrscheinlickeit, dass sich der Agent dorthin verirrt. Und desto größer wird die Wahrscheinlichkeit, dass er den Ausgang erreicht.

c) Relation zwischen den Ausgängen:

Um die Simulation für verschiedene Situationen zu testen, wurde sie jeweils mit dem Ost-Ausgang, dem Süd-Ausgang oder beiden Ausgängen gleichzeitig abgespielt. Betrachtet man die Diagramme genauer, fällt auf dass der Prozentsatz der entkommenen Agenten bei der Simulation mit dem Ost-Ausgang addiert mit dem Prozentsatz der entkommenen Agenten bei der Simulation mit dem Süd-Ausgang in etwa den Prozentsatz der entkommenen Agenten bei der Simulation mit beiden Ausgängen ergibt.

d) Relation zwischen den Simulationen

Vergleicht man die Werte, die man aus dem Diagramm mit der Simulation mit zehn Agenten in der ersten Variation entnehmen kann mit denen, die man aus der Simulation mit zehn Agenten in der zweiten Variation entnehmen kann, so fällt auf, dass im Schnitt in der zweiten Variante ca. 20% mehr Agenten gerettet werden als in der ersten Variante.

e) Wie lange braucht man, um alle, bzw. möglichst viele Agenten zu retten?

Diese Frage kann nicht vollständig beantwortet, sondern nur theoretisch abgehandelt werden. Die Beantwortung hängt von diversen Faktoren ab, wie z.B. der Platzierung des Ausgangs, der Anzahl der Agenten und der Anzahl der Ausgänge.

Je schneller die Agenten entkommen, desto schneller breitet sich das Feuer aus.

Je mehr Agenten sich gleichzeitig in dem Raum aufhalten, desto langsamer breitet sich das Feuer aus.

Je mehr Agenten sich gleichzeitig im Raum befinden, desto höher wird die Wahrscheinlichkeit, dass eine größere Anzahl an Agenten gerettet wird, jedoch bleibt der Prozentsatz der geretteten Agenten in etwa gleich.

Anhang: Programm zur Fluchtwegsimulation

Klasse Agent

```
1  import java.awt.Color;
2  import java.awt.Graphics;
3  import java.applet.*;
4  import java.awt.*;
5
6  public class Agent extends Applet
7  {
8    int x = (int)((Math.random())*375+100);
9    int y = (int)((Math.random())*375+100);
10   boolean foundWayOut = false;
11   int richtung = (int) ((Math.random()) * 4 + 1);
12
13   // 2 getter x und y
14   public int returnX ()
15   {
16     return x;
17   }
18
19   public int returnY ()
20   {
21     return y;
22   }
23
24   public int returnRichtung ()
25   {
26     return richtung;
27   }
28 }
```

Klasse Variante I

```
1  import java.applet.*;
2  import java.awt.*;
3
4  public class Fluchtwegsimulation extends Applet implements
       Runnable {
5    int anzahl; // Anzahl der Agenten
6    int groesse; // Groesse des Raumes
7    int curFeuerX = 50; //Breite des Feuers
8    int radius; // Groesse der Agenten
9    double gerettet; // Anzahl der geretteten Agenten
10   int xMauer; // x-Koordinate, an der die Mauer anfaengt
```

```
11  int yMauer; // y-Koordinate, an der die Mauer anfaengt
12  int bMauer; // Breite der Mauer
13  int lMauer; // Laenge der Mauer
14  int count;
15  int min;
16  int sec;
17  int countSec;
18  boolean mauer;
19  boolean running;
20  int[][] Raum;
21  Agent AgentenArray[];
22
23  // Festlegen der Werte im Konstruktor
24  public Fluchtwegsimulation() {
25    anzahl = 50;
26    groesse = 500;
27    Raum = new int[groesse][groesse];
28    AgentenArray = new Agent[anzahl];
29    radius = 10;
30    mauer = false;
31    running = false;
32
33    // Erzeugen der Mauer
34        if (mauer == false) {
35          mauer = true;
36          xMauer = 400;
37          yMauer = 150;
38          lMauer = 100;
39          bMauer = 10;
40        }
41
42    // Speichern der Agenten im dafuer vorgesehenem Array
43    for (int i = 0; i < anzahl; i++) {
44      AgentenArray[i] = new Agent();
45    }
46
47
48  }
49
50  public void paint(Graphics g) {
51    setSize(550, 550);
52    g.fillRect(50, 50, 450, 450);
53    g.setColor(Color.lightGray);
54    g.fillRect(475, 250, 50, 30); // Ausgang 1
55    g.fillRect(300, 475, 30, 50); // Ausgang 2
56    g.drawOval(425, 215, 100, 100); // Radius r1
57    g.drawOval(263, 440, 100, 100); // Radius r2
```

```
58      g.drawRect(50, 50, curFeuerX+50, 450); // Abstand a
59      g.setColor(Color.green);
60      g.fillRect(350, 10, 60, 30); // Anzeige 3
61      g.setColor(Color.red);
62      g.fillRect(50, 50, curFeuerX-25, 450); // Feuer
63      g.fillArc(70, 70, curFeuerX, 450, curFeuerX+75, curFeuerX
            +50); // Feuer
64      g.fillArc(50, 50, curFeuerX, 450, curFeuerX, curFeuerX); //
             Feuer
65      g.fillArc(50, 280, curFeuerX, 450, curFeuerX, curFeuerX+50)
            ; // Feuer
66      g.fillRect(255, 10, 70, 30); // Anzeige 2
67      g.setColor(Color.gray);
68      g.fillRect(130, 10, 100, 30); // Anzeige 1
69      g.fillRect(xMauer, yMauer, lMauer, bMauer); // Mauer
70      g.setColor(Color.black);
71      g.drawString("Zeit", 140, 20); // Anzeige 1
72      g.drawString("im Raum", 265, 20); //Anzeige 2
73      g.drawString("gerettet", 360, 20); // Anzeige 3
74      g.drawString(min+":"+sec+" min", 140, 30); // Anzeige 1
75      g.drawString(anzahl-gerettet+" ", 265, 30); // Anzeige 2
76      g.drawString(gerettet/anzahl*100+"%", 360, 30);// Anzeige 3
77      g.setColor(Color.white);
78      g.fillRect(500, 50, 500, 500);
79      g.fillRect(50, 500, 500, 500);
80      g.fillRect(410, 10, 100, 30);
81
82      Fuellen(g);
83    }
84
85    // Erstellung des Agenten
86    public void Fuellen(Graphics g) {
87      for (int i = 0; i < anzahl; i++) {
88        if (!AgentenArray[i].foundWayOut) {
89          g.setColor(Color.blue);
90          g.fillOval(AgentenArray[i].returnX(),
91              AgentenArray[i].returnY(), radius, radius);
92        }
93      }
94    }
95
96    // Neuzuweisung der Richtung
97    public int neuberechnen() {
98      int r = (int) ((Math.random()) * 4 + 1);
99      return r;
100   }
101
```

```
102   public void start() {
103     Thread th = new Thread(this);
104     th.start();
105   }
106
107   // Variante 1: Der Agent geht in die entgegengesetzte
          Richtung, wenn er an
108   // eine Wand stoesst.
109   public void run() {
110     while (true) {
111       for (int i = 0; i < anzahl; i++) {
112         int a = AgentenArray[i].returnX();
113         int b = AgentenArray[i].returnY();
114         int richtung = AgentenArray[i].returnRichtung();
115         if (countSec == 20) {
116           sec++;
117           countSec = 0;
118         }
119         if (sec == 60) {
120           min++;
121           sec = 0;
122         }
123         // richtung 1 = Norden (x/y-radius)
124         if (richtung == 1) {
125           if (AgentenArray[i].foundWayOut == true) {
126             continue;
127           } else if (b - radius <= 50) {
128             richtung = 3;
129           } else if (Raum[a][b-radius] == 1) {
130             while (richtung == 1) {
131               neuberechnen();
132             }
133           } else if (a >= xMauer && b >= yMauer-5 && b <=
                yMauer) {
134             richtung = 4;
135           } else if (a >= xMauer && b >= yMauer+10 && b <=
                yMauer+15) {
136             while (richtung == 1) {
137               neuberechnen();
138             }
139           } else {
140             Raum[a][b - radius] = 1;
141             Raum[a][b] = 0;
142             b = b - radius;
143             repaint();
144             try {
145               Thread.sleep(50);
```

```
146             } catch (InterruptedException e) {
147             }
148             if (a <= curFeuerX+100) {
149               richtung = 2;
150             } else if (a > 435 && b > 220 && b < 310) {
151               AgentenArray[i].foundWayOut = true;
152               gerettet++;
153             } else if (a > 275 && b > 455 && a < 345) {
154               AgentenArray[i].foundWayOut = true;
155               gerettet++;
156             } else {
157               richtung = (int) ((Math.random()) * 4 + 1);
158             }
159           }
160           countSec++;
161         }
162         // richtung 2 = Osten (x+radius/y)
163         else if (richtung == 2) {
164           if (AgentenArray[i].foundWayOut == true) {
165           continue;
166           } else  if (a + radius >= 475) {
167             richtung = 4;
168           } else if (Raum[a + radius][b] == 1) {
169             while (richtung == 2) {
170               neuberechnen();
171             }
172           } else if (a >= xMauer && b >= yMauer-5 && b <=
                yMauer) {
173             richtung = 4;
174           } else if (a >= xMauer && b >= yMauer+10 && b <=
                yMauer+15) {
175             while (richtung == 1) {
176               neuberechnen();
177             }
178           } else {
179             Raum[a + radius][b] = 1;
180             Raum[a][b] = 0;
181             a = a + radius;
182             repaint();
183             try {
184               Thread.sleep(50);
185             } catch (InterruptedException e) {
186             }
187             if (a <= curFeuerX+100) {
188               richtung = 2;
189             } else if (a > 435 && b > 220 && b < 310) {
190               AgentenArray[i].foundWayOut = true;
```

```
191               gerettet++;
192             } else if (a > 275 && b > 455 && a < 345) {
193               AgentenArray[i].foundWayOut = true;
194               gerettet++;
195             } else if (a == 475 && b < 220 || a == 475 && b >
                    310) {
196               richtung = 4;
197             } else {
198               richtung = (int) ((Math.random()) * 4 + 1);
199             }
200           }
201           countSec++;
202         }
203         // richtung 3 = Sueden (x/y+radius)
204         else if (richtung == 3) {
205           if (AgentenArray[i].foundWayOut == true) {
206             continue;
207           } else if (b + radius >= 475) {
208             richtung = 1;
209           } else if (Raum[a][b + radius] == 1) {
210             while (richtung == 3) {
211               neuberechnen();
212             }
213           } else if (a >= xMauer && b >= yMauer-5 && b <=
                    yMauer) {
214             richtung = 4;
215           } else if (a >= xMauer && b >= yMauer+10 && b <=
                    yMauer+15) {
216             while (richtung == 1) {
217               neuberechnen();
218             }
219           } else {
220             Raum[a][b + radius] = 1;
221             Raum[a][b] = 0;
222             b = b + radius;
223             repaint();
224             try {
225               Thread.sleep(50);
226             } catch (InterruptedException e) {
227             }
228             if (a <= curFeuerX+100) {
229               richtung = 2;
230             } else if (b == 475) {
231               richtung = 1;
232             } else if (a > 435 && b > 220 && b < 310) {
233               AgentenArray[i].foundWayOut = true;
234               gerettet++;
```

```
235             } else if (a > 275 && b > 455 && a < 345) {
236               AgentenArray[i].foundWayOut = true;
237               gerettet++;
238             }else {
239               richtung = (int) ((Math.random()) * 4 + 1);
240             }
241           }
242           countSec++;
243         }
244         // richtung 4 = Westen (x-radius/y)
245         else if (richtung == 4) {
246           if (AgentenArray[i].foundWayOut == true) {
247             continue;
248           } else if (Raum[a - radius][b] == 1) {
249             while (richtung == 4) {
250               neuberechnen();
251             }
252           } else if (a >= xMauer && b >= yMauer-5 && b <=
                yMauer) {
253             richtung = 4;
254           } else if (a >= xMauer && b >= yMauer+10 && b <=
                yMauer+15) {
255             while (richtung == 1) {
256               neuberechnen();
257             }
258           } else {
259             Raum[a - radius][b] = 1;
260             Raum[a][b] = 0;
261             a = a - radius;
262             repaint();
263             try {
264               Thread.sleep(50);
265             } catch (InterruptedException e) {
266             }
267             if (a <= curFeuerX+100) {
268               richtung = 2;
269             } else if (a > 275 && b > 455 && a < 345) {
270               AgentenArray[i].foundWayOut = true;
271               gerettet++;
272             } else {
273               richtung = (int) ((Math.random()) * 4 + 1);
274             }
275           }
276           countSec++;
277         }
278         AgentenArray[i].x = a;
279         AgentenArray[i].y = b;
```

```
280           AgentenArray[i].richtung = richtung;
281         }// for
282         count++;
283         if(count > 10) {
284           count = 1;
285           curFeuerX += 10;
286         }
287         // stoppt die Simulation
288         if (curFeuerX+50 >= 475) {
289           running = false;
290           break;
291         }
292       }
293     }
294
295     public static void main(String[] args) {
296
297     }
298 }
```

Klasse Variante II

```
1  import java.applet.*;
2  import java.awt.*;
3
4  public class FluchtwegsimulationV2 extends Applet implements
       Runnable {
5    int anzahl; // Anzahl der Agenten
6    int groesse; // Groesse des Raumes
7    int curFeuerX = 50; //Breite des Feuers
8    int radius; // Groesse der Agenten
9    double gerettet; // Anzahl der geretteten Agenten
10   int richtungalt;
11   int count;
12   int min;
13   int sec;
14   int countSec;
15   boolean running;
16   int[][] Raum;
17   Agent AgentenArray[];
18
19   // Festlegen der Werte im Konstruktor
20   public FluchtwegsimulationV2() {
21     anzahl = 10;
22     groesse = 500;
23     Raum = new int[groesse][groesse];
24     AgentenArray = new Agent[anzahl];
```

```
25      radius = 10;
26      gerettet = 0;
27      min = 0;
28      sec = 0;
29      running = true;
30
31      // Speichern der Agenten in dem dafuer vorgesehenem Array
32      for (int i = 0; i < anzahl; i++) {
33        AgentenArray[i] = new Agent();
34      }
35
36      // Beschriften der Mauer mit der Ziffer 2
37      for (int x = 50; x < groesse; x++) {
38        for (int y = 50; y >= 55; y++) {
39          Raum[x][y] = 2;
40        }
41        for (int z = 471; z <= 475; z++) {
42          Raum[x][z] = 2;
43        }
44      }
45
46      for (int l = 50; l <= 475; l++) {
47        for (int m = 471; m <= 475; m++) {
48          Raum[m][l] = 2;
49        }
50      }
51    }
52
53    public void paint(Graphics g) {
54      setSize(550, 550);
55      g.fillRect(50, 50, 450, 450);
56      g.setColor(Color.lightGray);
57      g.fillRect(475, 250, 50, 30); // Ausgang 1
58      g.fillRect(300, 475, 30, 50); // Ausgang 2
59      g.drawOval(425, 215, 100, 100); // Radius r1
60      g.drawOval(263, 440, 100, 100); // Radius r2
61      g.drawRect(50, 50, curFeuerX+50, 450); // Abstand a
62      g.setColor(Color.green);
63      g.fillRect(350, 10, 60, 30); // Anzeige 3
64      g.setColor(Color.red);
65      g.fillRect(50, 50, curFeuerX-25, 450); // Feuer
66      g.fillArc(70, 70, curFeuerX, 450, curFeuerX+75, curFeuerX
            +50); // Feuer
67      g.fillArc(50, 50, curFeuerX, 450, curFeuerX, curFeuerX); //
             Feuer
68      g.fillArc(50, 280, curFeuerX, 450, curFeuerX, curFeuerX+50)
            ; // Feuer
```

```
69      g.fillRect(255, 10, 70, 30); // Anzeige 2
70      g.setColor(Color.gray);
71      g.fillRect(130, 10, 100, 30); // Anzeige 1
72      g.setColor(Color.black);
73      g.drawString("Zeit", 140, 20); // Anzeige 1
74      g.drawString("im Raum", 265, 20); //Anzeige 2
75      g.drawString("gerettet", 360, 20); // Anzeige 3
76      g.drawString(min+":"+sec+" min", 140, 30); // Anzeige 1
77      g.drawString(anzahl-gerettet+" ", 265, 30); // Anzeige 2
78      g.drawString(gerettet/anzahl*100+"% ", 360, 30);// Anzeige
            3
79      g.setColor(Color.white);
80      g.fillRect(500, 50, 500, 500);
81      g.fillRect(50, 500, 500, 500);
82      g.fillRect(410, 10, 100, 30);
83
84      Fuellen(g);
85    }
86
87    // Erstellung des Agenten
88    public void Fuellen(Graphics g) {
89      for (int i = 0; i < anzahl; i++) {
90        if (!AgentenArray[i].foundWayOut) {
91          g.setColor(Color.blue);
92          g.fillOval(AgentenArray[i].returnX(),
93              AgentenArray[i].returnY(), radius, radius);
94        }
95      }
96    }
97
98    // Neuzuweisung der Richtung
99    public int neuberechnen() {
100     int r = (int) ((Math.random()) * 4 + 1);
101     return r;
102   }
103
104   public void start() {
105     Thread th = new Thread(this);
106     th.start();
107   }
108
109   // Variante 2: Der Agent geht an der Wand weiter
110   public void run () {
111     while (true) {
112       for (int i=0; i<anzahl; i++) {
113         int a = AgentenArray[i].returnX();
114         int b = AgentenArray[i].returnY();
```

```
115         int richtung = AgentenArray[i].returnRichtung();
116         richtungalt = richtung;
117         if (countSec == 20) {
118           sec++;
119           countSec = 0;
120         }
121         if (sec == 60) {
122           min++;
123           sec = 0;
124         }
125
126         // richtung 1 = Norden (x/y-radius)
127         if (richtung == 1) {
128           if (AgentenArray[i].foundWayOut == true) {
129             continue;
130           } else if (Raum[a][b] == 2) {
131             if (richtungalt == 1) {
132               if (b-radius < 50) {
133                 neuberechnen();
134                 while (richtung == 3 || richtung == 1) {
135                   richtung = (int)((Math.random())*4+1);
136                   if (richtung == 2) {
137                     if (Raum[a+radius][b] == 1) {
138                       richtung = 4;
139                     } else if (a >= 465 && b <= 60) {
140                       if (Raum[a][b+radius] == 1) {
141                         richtung = 4;
142                       } else {
143                         Raum[a][b+radius] = 1;
144                         Raum[a][b] = 2;
145                         b = b+radius;
146                         repaint();
147                         try {
148                         Thread.sleep(50);
149                         } catch (InterruptedException e) {}
150                         richtung = 3;
151                       }
152                     } else if (a >= 471 && b >= 471) {
153                       if (Raum[a][b-radius] == 1) {
154                         richtung = 4;
155                       } else {
156                         Raum[a][b-radius] = 1;
157                         Raum[a][b] = 2;
158                         b = b-radius;
159                         repaint();
160                         try {
161                           Thread.sleep(50);
```

```
162                     } catch (InterruptedException e) {}
163                     richtung = 1;
164                   }
165                 } else if (a > 435 && b > 220 && b < 310) {
166                   AgentenArray[i].foundWayOut = true;
167                   gerettet++;
168                 } else if (a > 275 && b > 455 && a < 345) {
169                   AgentenArray[i].foundWayOut = true;
170                   gerettet++;
171                 } else if (Raum[a+radius][b] == 2) {
172                   Raum[a+radius][b] = 1;
173                   Raum[a][b] = 2;
174                   a = a+radius;
175                   repaint();
176                   try {
177                     Thread.sleep(50);
178                   } catch (InterruptedException e) {}
179                 }
180               } else if (richtung == 4) {
181                 if (Raum[a-radius][b] == 1) {
182                   richtung = 2;
183                 } else if (a >= 465 && b <= 60) {
184                   if (Raum[a][b+radius] == 1) {
185                     richtung = 4;
186                   } else {
187                       Raum[a][b+radius] = 1;
188                       Raum[a][b] = 2;
189                       b = b+radius;
190                       repaint();
191                       try {
192                         Thread.sleep(50);
193                       } catch (InterruptedException e) {}
194                       richtung = 3;
195                   }
196                 } else if (a >= 471 && b >= 471) {
197                   if (Raum[a][b-radius] == 1) {
198                     richtung = 4;
199                   } else {
200                     Raum[a][b-radius] = 1;
201                     Raum[a][b] = 2;
202                     b = b-radius;
203                     repaint();
204                     try {
205                       Thread.sleep(50);
206                     } catch (InterruptedException e) {}
207                     richtung = 1;
208                   }
```

```
209                     }else if (a <= curFeuerX+100) {
210                       richtung = 2;
211                     } else if (a > 435 && b > 220 && b < 310) {
212                       AgentenArray[i].foundWayOut = true;
213                       gerettet++;
214                     } else if (a > 275 && b > 455 && a < 345) {
215                       AgentenArray[i].foundWayOut = true;
216                       gerettet++;
217                     } else if (Raum[a-radius][b] == 2){
218                       Raum[a-radius][b] = 1;
219                       Raum[a][b] = 2;
220                       a = a-radius;
221                       repaint();
222                       try {
223                         Thread.sleep(50);
224                       } catch (InterruptedException e) {}
225                     }
226                   }
227                 }
228               } else {
229                 if (Raum[a][b-radius] == 1) {
230                   richtung = 3;
231                 } else {
232                   Raum[a][b-radius] = 1;
233                   Raum[a][b] = 2;
234                   b = b-radius;
235                   repaint();
236                   try {
237                     Thread.sleep(50);
238                   } catch (InterruptedException e) {}
239                 }
240               }
241             } else if (richtungalt == 2) {
242               richtung = 2;
243             } else if (richtungalt == 4) {
244               richtung = 4;
245             }
246           } else if (Raum[a][b-radius] == 1) {
247             while (richtung == 1) {
248               neuberechnen();
249             }
250           } else {
251             Raum[a][b-radius] = 1;
252             Raum[a][b] = 0;
253             b = b-radius;
254             repaint();
255             try {
```

```
256               Thread.sleep(50);
257             } catch (InterruptedException e) {}
258             if (a <= curFeuerX+100) {
259               richtung = 2;
260             } else if (a > 435 && b > 220 && b < 310) {
261               AgentenArray[i].foundWayOut = true;
262               gerettet++;
263             } else if (a > 275 && b > 455 && a < 345) {
264               AgentenArray[i].foundWayOut = true;
265               gerettet++;
266             } else {
267               richtung = (int) ((Math.random()) * 4 + 1);
268             }
269           }
270           richtungalt = richtung;
271           countSec++;
272
273         // richtung 2 = Osten (x+radius/y)
274         } else if (richtung == 2) {
275           if (AgentenArray[i].foundWayOut == true) {
276             continue;
277           } else if (Raum[a][b] == 2) {
278             if (richtungalt == 2) {
279               if (a+radius > 475) {
280                 neuberechnen();
281                 while (richtung == 2 || richtung == 4) {
282                   richtung = (int)((Math.random())*4+1);
283                   if (richtung == 1) {
284                     if (Raum[a][b-radius] == 1) {
285                       richtung = 3;
286                     } else if (a >= 465 && b <= 60) {
287                       if (Raum[a-radius][b] == 1) {
288                         richtung = 3;
289                       } else {
290                         Raum[a-radius][b] = 1;
291                         Raum[a][b] = 2;
292                         a = a-radius;
293                         repaint();
294                         try {
295                           Thread.sleep(50);
296                         } catch (InterruptedException e) {}
297                         richtung = 4;
298                       }
299                     } else if (a >= 465 && b >= 465) {
300                       if (Raum[a+radius][b] == 1) {
301                         richtung = 4;
302                       } else {
```

```
303                     Raum[a+radius][b] = 1;
304                     Raum[a][b] = 2;
305                     a = a+radius;
306                     repaint();
307                     try {
308                       Thread.sleep(50);
309                     } catch (InterruptedException e) {}
310                     richtung = 2;
311                   }
312                 } else if (a > 435 && b > 220 && b < 310) {
313                   AgentenArray[i].foundWayOut = true;
314                   gerettet++;
315                 } else if (a > 275 && b > 455 && a < 345) {
316                   AgentenArray[i].foundWayOut = true;
317                   gerettet++;
318                 } else if (Raum[a][b-radius] == 2) {
319                   Raum[a][b-radius] = 1;
320                   Raum[a][b] = 2;
321                   b = b-radius;
322                   repaint();
323                   try {
324                     Thread.sleep(50);
325                   } catch (InterruptedException e) {}
326                 }
327               } else if (richtung == 3) {
328                 if (Raum[a][b+radius] == 1) {
329                   richtung = 1;
330                 } else if (a >= 465 && b >= 465) {
331                   if (Raum[a-radius][b] == 1) {
332                     richtung = 1;
333                   } else {
334                     Raum[a-radius][b] = 1;
335                     Raum[a][b] = 2;
336                     a = a-radius;
337                     repaint();
338                     try {
339                       Thread.sleep(50);
340                     } catch (InterruptedException e) {}
341                     richtung = 4;
342                   }
343                 } else if (a >= 465 && b <= 60) {
344                   if (Raum[a][b+radius] == 1) {
345                     richtung = 4;
346                   } else {
347                     Raum[a][b+radius] = 1;
348                     Raum[a][b] = 2;
349                     b = b+radius;
```

```
350                         try {
351                           Thread.sleep(50);
352                         } catch (InterruptedException e) {}
353                       }
354                     } else if (a > 435 && b > 220 && b < 310) {
355                       AgentenArray[i].foundWayOut = true;
356                       gerettet++;
357                     } else if (a > 275 && b > 455 && a < 345) {
358                       AgentenArray[i].foundWayOut = true;
359                       gerettet++;
360                     } else if (Raum[a][b+radius] == 2){
361                       Raum[a][b+radius] = 1;
362                       Raum[a][b] = 2;
363                       b = b+radius;
364                       repaint();
365                       try {
366                         Thread.sleep(50);
367                       } catch (InterruptedException e) {}
368                     }
369                   }
370                 }
371               } else if (Raum[a+radius][b] == 1) {
372                 richtung = 4;
373               } else {
374                 Raum[a+radius][b] = 1;
375                 Raum[a][b] = 2;
376                 a = a+radius;
377                 repaint();
378                 try {
379                   Thread.sleep(50);
380                 } catch (InterruptedException e) {}
381               }
382             } else if (richtungalt == 1) {
383               richtung = 1;
384             } else if (richtungalt == 3) {
385               richtung = 3;
386             }
387           } else if (Raum[a+radius][b] == 1) {
388             while (richtung == 2) {
389               neuberechnen();
390             }
391           } else {
392             Raum[a+radius][b] = 1;
393             Raum[a][b] = 0;
394             a = a+radius;
395             repaint();
396             try {
```

```
397               Thread.sleep(50);
398             } catch (InterruptedException e) {}
399             if (a <= curFeuerX+100) {
400               richtung = 2;
401             } else if (a > 435 && b > 220 && b < 310) {
402               AgentenArray[i].foundWayOut = true;
403               gerettet++;
404             } else if (a > 275 && b > 455 && a < 345) {
405               AgentenArray[i].foundWayOut = true;
406               gerettet++;
407             } else {
408               richtung = (int) ((Math.random()) * 4 + 1);
409             }
410           }
411           richtungalt = richtung;
412           countSec++;
413
414         // Sueden (x/y+radius)
415         } else if (richtung == 3) {
416           if (AgentenArray[i].foundWayOut == true) {
417             continue;
418           } else if (Raum[a][b] == 2) {
419             if (richtungalt == 3) {
420               if (b+radius > 475) {
421                 neuberechnen();
422                 while (richtung == 3 || richtung == 1) {
423                   richtung = (int)((Math.random())*4+1);
424                   if (richtung == 2) {
425                     if (Raum[a+radius][b] == 1) {
426                       richtung = 4;
427                     } else if (a >= 465 && b <= 60) {
428                       if (Raum[a][b+radius] == 1) {
429                         richtung = 4;
430                       } else {
431                         Raum[a][b+radius] = 1;
432                         Raum[a][b] = 2;
433                         b = b+radius;
434                         repaint();
435                         try {
436                           Thread.sleep(50);
437                         } catch (InterruptedException e) {}
438                         richtung = 3;
439                       }
440                     } else if (a >= 465 && b >= 465) {
441                       if (Raum[a][b-radius] == 1) {
442                         richtung = 4;
443                       } else {
```

```
444                     Raum[a][b-radius] = 1;
445                     Raum[a][b] = 2;
446                     b = b-radius;
447                     repaint();
448                     try {
449                       Thread.sleep(50);
450                     } catch (InterruptedException e) {}
451                     richtung = 1;
452                   }
453                 } else if (a > 435 && b > 220 && b < 310) {
454                   AgentenArray[i].foundWayOut = true;
455                   gerettet++;
456                 } else if (a > 275 && b > 455 && a < 345) {
457                   AgentenArray[i].foundWayOut = true;
458                   gerettet++;
459                 } else if (Raum[a+radius][b] == 2) {
460                   Raum[a+radius][b] = 1;
461                   Raum[a][b] = 2;
462                   a = a+radius;
463                   repaint();
464                   try {
465                     Thread.sleep(50);
466                   } catch (InterruptedException e) {}
467                 }
468               } else if (richtung == 4) {
469                 if (Raum[a-radius][b] == 1) {
470                   richtung = 2;
471                 } else if (a >= 465 && b <= 60) {
472                   if (Raum[a-radius][b] == 1) {
473                     richtung = 3;
474                   } else {
475                     Raum[a-radius][b] = 1;
476                     Raum[a][b] = 2;
477                     a = a-radius;
478                     repaint();
479                     try {
480                       Thread.sleep(50);
481                     } catch (InterruptedException e) {}
482                   }
483                 } else if (a >= 465 && b >= 465) {
484                   if (Raum[a-radius][b] == 1) {
485                     richtung = 1;
486                   } else {
487                     Raum[a-radius][b] = 1;
488                     Raum[a][b] = 2;
489                     a = a-radius;
490                     repaint();
```

```
491                         try {
492                           Thread.sleep(50);
493                         } catch (InterruptedException e) {}
494                       }
495                     } else if (a <= curFeuerX) {
496                       richtung = 2;
497                     } else if (a > 435 && b > 220 && b < 310) {
498                       AgentenArray[i].foundWayOut = true;
499                       gerettet++;
500                     } else if (a > 275 && b > 455 && a < 345) {
501                       AgentenArray[i].foundWayOut = true;
502                       gerettet++;
503                     } else if (Raum[a-radius][b] == 2){
504                       Raum[a-radius][b] = 1;
505                       Raum[a][b] = 2;
506                       a = a-radius;
507                       repaint();
508                       try {
509                         Thread.sleep(50);
510                       } catch (InterruptedException e) {}
511                     }
512                   }
513                 }
514               } else if (Raum[a][b+radius] == 1) {
515                 richtung = 1;
516               } else {
517                 Raum[a][b+radius] = 1;
518                 Raum[a][b] = 2;
519                 b = b+radius;
520                 repaint();
521                 try {
522                   Thread.sleep(50);
523                 } catch (InterruptedException e) {}
524               }
525             } else if (richtungalt == 2) {
526               richtung = 2;
527             } else if (richtungalt == 4) {
528               richtung = 4;
529             }
530           } else if (Raum[a][b+radius] == 1) {
531             while (richtung == 3) {
532               neuberechnen();
533             }
534           } else {
535             Raum[a][b+radius] = 1;
536             Raum[a][b] = 0;
537             b = b+radius;
```

```
538             repaint();
539             try {
540               Thread.sleep(50);
541             } catch (InterruptedException e) {}
542             if (a <= curFeuerX+100) {
543               richtung = 2;
544             } else if (a > 435 && b > 220 && b < 310) {
545               AgentenArray[i].foundWayOut = true;
546               gerettet++;
547             } else if (a > 275 && b > 455 && a < 345) {
548               AgentenArray[i].foundWayOut = true;
549               gerettet++;
550             } else {
551               richtung = (int) ((Math.random()) * 4 + 1);
552             }
553           }
554           richtungalt = richtung;
555           countSec++;
556
557         // Westen (x-radius/y)
558         } else if (richtung == 4) {
559           if (AgentenArray[i].foundWayOut == true) {
560             continue;
561           } else if (Raum[a][b] == 2) {
562             if (richtungalt == 4) {
563               if (a <= curFeuerX + 100) {
564               richtung = 2;
565               } else if (a > 435 && b > 220 && b < 310) {
566                 AgentenArray[i].foundWayOut = true;
567                 gerettet++;
568               } else if (a > 275 && b > 455 && a < 345) {
569                 AgentenArray[i].foundWayOut = true;
570                 gerettet++;
571               } else if (Raum[a-radius][b] == 1) {
572                 richtung = 2;
573               } else {
574                 Raum[a-radius][b] = 1;
575                 Raum[a][b] = 2;
576                 a = a-radius;
577                 repaint();
578                 try {
579                   Thread.sleep(50);
580                 } catch (InterruptedException e) {}
581               }
582             } else if (a <= curFeuerX + 100) {
583               richtung = 2;
584               } else if (a >= 465 && b <= 60) {
```

```
585                 if (Raum[a-radius][b] == 1) {
586                   richtung = 3;
587                 } else {
588                   Raum[a-radius][b] = 1;
589                   Raum[a][b] = 2;
590                   a = a-radius;
591                   repaint();
592                   try {
593                     Thread.sleep(50);
594                   } catch (InterruptedException e) {}
595                 }
596               } else if (a >= 465 && b >= 465) {
597                 if (Raum[a-radius][b] == 1) {
598                   richtung = 1;
599                 } else {
600                   Raum[a-radius][b] = 1;
601                   Raum[a][b] = 2;
602                   a = a-radius;
603                   repaint();
604                   try {
605                     Thread.sleep(50);
606                   } catch (InterruptedException e) {}
607                 }
608               } else if (a > 435 && b > 220 && b < 310) {
609                 AgentenArray[i].foundWayOut = true;
610                 gerettet++;
611               } else if (a > 275 && b > 455 && a < 345) {
612                 AgentenArray[i].foundWayOut = true;
613                 gerettet++;
614               }
615             } else if (Raum[a-radius][b] == 1) {
616               while (richtung == 4) {
617               neuberechnen();
618               }
619             } else {
620               Raum[a-radius][b] = 1;
621               Raum[a][b] = 0;
622               a = a-radius;
623               repaint();
624               try {
625                 Thread.sleep(50);
626               } catch (InterruptedException e) {}
627               if (a > 435 && b > 220 && b < 310) {
628                 AgentenArray[i].foundWayOut = true;
629                 gerettet++;
630               } else if (a > 275 && b > 455 && a < 345) {
631                 AgentenArray[i].foundWayOut = true;
```

```
632                 gerettet++;
633               } else {
634                 richtung = (int) ((Math.random()) * 4 + 1);
635               }
636             }
637           richtungalt = richtung;
638           countSec++;
639         }
640         AgentenArray[i].x = a;
641         AgentenArray[i].y = b;
642         AgentenArray[i].richtung = richtung;
643       }
644       count++;
645       if(count > 10) {
646         count = 1;
647         curFeuerX += 10;
648       }
649       // stoppt die Simulation
650       if (curFeuerX+50 >= 475) {
651         stop();
652         running = false;
653       }
654     }
655   }
656
657
658   public static void main(String[] args) {
659
660   }
661 }
```

Literaturverzeichnis

[Ber93] Bertelsmann, Lexikon-Institut: Die Grosse Bertelsmann Lexikothek. In: *Bertelsmann Lexikon Band 13*. 1993, S. 234

[BfG14] BfGA: *Fluchtweg - Definition*. `http://www.bfga.de/arbeitsschutz-lexikon-von-a-bis-z/fachbegriffe-c-i/fluchtweg`. Version: 2014

[HFV00] Helbing, Dirk ; Farkas, Illés ; Vicsek, Tamás: Simulating dynamical features of escape panic. In: *Letters to Nature* 407 (2000)

[Joa14] Joanna: *Randomisierung/fremdwort.de - Was ist Randomisierung - Definition, Bedeutung, Herkunft*. `http://www.fremdwort.de/suchen/bedeutung/Randomisierung`. Version: 2014

[SF14] Springer Fachmedien, Wiesbaden G.: *Definition »Agent« | Gabler Wirtschaftslexikon*. `http://wirtschaftslexikon.gabler.de/Definition/agent.html#definition`. Version: 2014

Ich erkläre hiermit, dass ich die Seminararbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

…....., den …....
Ort Datum

…....
Unterschrift des Verfassers