Geosensornetze
WS 2013/2014

Hausarbeit von
Andre Lehnert und Marcell Salvage

12. Februar 2014

Inhaltsverzeichnis

1 Einführung **1**
1.1 Aufgabenbeschreibung 1

2 Simulationsumgebung **3**
2.1 Benutzerschnittstelle 3
2.1.1 Benutzerevents 3
2.1.2 Generelle Parameter 5
2.1.3 Personen–Parameter 6
2.1.4 Event–Parameter 7
2.1.5 Notausgang–Parameter 7
2.1.6 Multilateration localization–Parameter 8
2.2 Personen 8
2.2.1 Lebenszyklus 9
2.2.2 Bewegungsmodell 9
2.2.3 Kommunikationsmodell 12
2.3 Notausgänge 14
2.3.1 Lebenszyklus 14
2.3.2 Kommunikationsmodell 14
2.4 Gefahrensituationen 16
2.4.1 Lebenszyklus 16
2.5 Patches 17
2.5.1 Implizite Zustände 17
2.6 Ressourcen der Simulationsumgebung 19

3 Algorithmik **21**
3.1 Multilateration Lokalisierung 21
3.2 Zellulärer Automat 22
3.2.1 Orientierung der Personen bei der Flucht 25

4 Evaluation **26**
4.1 Lokalisierung 26
4.2 Evakuierungsdauer 29

4.3 Effizienz 29
4.4 Fazit 29
4.5 Ausblick 29

Literaturverzeichnis **i**

1 Einführung

Mit Hilfe der NetLogo-Simulationsumgebung [3] wird eine dynamische Evakuierung von Gebäuden implementiert. Dazu werden die Grundrisse der Gebäude oder Etage in die Simulationsumgebung geladen. Diese dient als Grundlage für die Platzierung von Personen, Gefahrenevents und Notausgängen.

Auf der Flucht vor Gefahrenevents werden die Personen von mobilen Geräten unterstützt, die zur Warnung anderer Personen und zur Lokalisierung der Notausgänge dienen.

1.1 Aufgabenbeschreibung

Die Aufgabe besteht in der Umsetzung einer geeigneten Simulationsumgebung (siehe Kapitel 2). Auf deren Basis Algorithmen zur Lokalisierung und Bestimmung eines Fluchtweges zu den Notausgängen entwickelt werden (siehe Kapitel 3). Schließlich wird eine Evaluation der der Algorithmen in Punkto Effizienz und Zuverlässigkeit durchgeführt und Reflektiert (siehe Kapitel 4).

Personen werden in der NetLogo-Umgebung als Agenten realisiert, die sich nach dem Bewegungsmodell (siehe Abschnitt 2.2.2) innerhalb des Grundrisses bewegen. Die initiale Platzierung geschieht zufällig, analog zu der Platzierung der Gefahrenevents. Personen besitzen die Fähigkeit diese Gefahrenevents in ihrer Umgebung wahrzunehmen und als Gefahrensituation zu deuten. Der Personen versuchen daraufhin den besten Weg zu einem Notausgang zu finden und benachbarte Personen dabei über ihre mobilen Geräte zu warnen.

Als Gefahrensituationen (siehe Abschnitt 2.4) zählt eine gewisse Anzahl von Giftgasbomben mit eingebautem Zeitzünder, die je ein Gefahrenevent darstellen. Die freigesetzten Gasmengen sind regulierbar und breiten sich innerhalb des freien Raumes aus.

Zur Evakuierung der Personen aus dem Gefahrenevent werden Notausgänge (siehe Abschnitt 2.3) in dem Grundriss platziert, deren Position sich während der Simulation nicht ändert, sogenannte *anchor notes*.

Eine feste Position ist notwendig zur Realisierung der dezentralen Lokalisierungs-

algorithmen, die auf den mobilen Geräten der Personen aktiv sind und bei einer dynamischen Evakuierung assistieren. Durch eine lokal eingeschränkte Kommunikationsfähigkeit (siehe Abschnitt 2.2.3) werden Informationen über die Passierbarkeit der Notausgänge an die mobilen Geräte verteilt. Dies ermöglicht die sichere Evakuierung, falls beispielsweise das Giftgas einen Notausgang erreicht hat oder die Fluchtwege blockiert sind.

2 Simulationsumgebung

Das Kapitel der Simulationsumgebung befasst sich mit den Eingabemöglichkeiten zur Anpassung der Simulationsparameter, sowie der konkreten Umsetzung der simulierten Welt, mit der Logik für die Notausgänge, der Personen, der Gefahrensituationen und dem Kommunikationsmodell.

Die Ausgangsbasis für die verwendeten Modelle und Protokolle stammen aus den erstellten Übungen zur Vorlesung Gensensornetze, sie wurden jedoch auf die Anforderungen der Hausarbeit adaptiert.

2.1 Benutzerschnittstelle

In diesem Abschnitt werden die Interaktions- und Konfigurationsmöglichkeiten mit der Simulationsumgebung erläutert. Abbildung 2.1 stellt einen Ausschnitt des grafischen Interfaces von NetLogo da, anhand dessen die Erklärungen in den folgenden Abschnitten besser einzuordnen sind.

2.1.1 Benutzerevents

Zum Auslösen von Benutzerevents werden Buttons verwendet. Zu diesen zählen `setup`, `reset` und `go`, die hier kurz beschrieben werden.

setup-Button

Die Betätigung des `setup`-Buttons ruft eine Folge von Setup und Initialisierungsschritten auf. Zu Beginn wird die Simulationswelt erstellt. Dazu wird der Parameter `inputFile` (siehe Abschnitt 2.1.2) zum Einbinden einer Bild-Datei und das Mapping der Pixel-Farbwerte auf die passend skalierte NetLogo-Welt gemappt. Das Ergebnis ist der gewählte Grundriss, bei dem jedes Patch einen Farbwert erhalten hat.

Dieser Farbwert ist essentiell für den nachfolgenden Schritt in der `setup-patches`-Methode. Diese legt den Initialzustand jedes Patches fest (siehe Abschnitt 2.5).

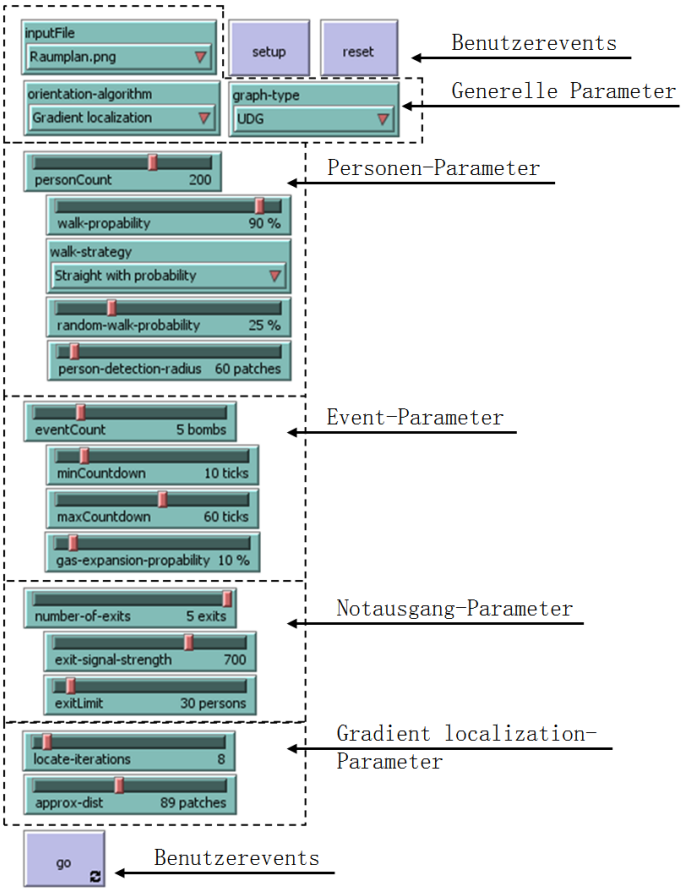

Abbildung 2.1: Grafische Oberfläche

Generell wird auf Grund des Grundrisses zwischen leerem Raum (weiß) und einer Wand unterschieden (schwarz).

Wände spielen auch bei der Initialisierung und Platzierung der Notausgänge eine wichtige Rolle, die mit der `setup-exits`-Methode realisiert sind. Entsprechend der Aufgabenstellung werden Notausgänge auf einem abstrakten Grundriss, ohne Wände, zufällig in der Welt platziert. Für alle anderen Grundrisse mit Wänden wurde auf statische vordefinierte Positionen für Notausgänge gesetzt.

Bei der Initialisierung wird zudem die lokale Konstante der maximalen Signalreichweite für den Orientierungsalgorithmus *cellular automaton* mit dem Parameter `exit-signal-strength` aus Abschnitt 2.1.5 gesetzt und alle Notausgänge in den Zustand *INIT* versetzt (vgl. Abschnitt 2.3).

Im Anschluss wird der *cellular automaton*-Algorithmus (siehe Abschnitt 3.2) ausgeführt, um die Patches mit Signalqualitätsinformationen auszustatten, damit die Personen den optimalen Fluchtweg bei einer Gefahrensituation nutzen.

Nachdem die Simulationswelt mit den statischen Elementen vorbereitet wurde, werden die Personen initialisiert und platziert. Dies geschieht mit der Methode `setup-persons`. Hier wird eine definierte Anzahl von Personen erstellt (siehe Abschnitt 2.1.3, die auf einem abstrakten Grundriss zufällig und auf allen anderen zufällig mit einer Wand-Detektion platziert werden. Die Personen befinden sich nun im Zustand *INIT* (vgl. Abschnitt 2.2.2).

Nach der Platzierung wird einmalig ein Kommunikationsgraph zwischen den Personen und den Notausgängen erstellt, damit der Nutzer ggf. den Graph-Typen oder die Kantenlänge anpassen kann (siehe Abschnitt 2.1.3).

Als letzter Schritt folgt die Initialisierung und Platzierung der Gefahrenevents mittels `setup-events`. Die analog zu den Personen auf dem abstrakten Grundriss zufällig und bei allen anderen Grundrissen mit Wand-Detektion platziert werden. Danach befinden sich alle Gefahrenevents im Zustand *INIT* (vgl. Abschnitt 2.4).

reset-Button

Mit dem `reset`-Button werden alle definierten Parameter des letzten Setups wiederhergestellt und die Personen auf ihre ursprüngliche Position zurückgesetzt. Ein fluten des Grundrisses ist nicht erforderlich und beschleunigt das durchführen von Messreihenreihen. Zudem bietet es die Möglichkeit den Kommunikationsgraphen der Personen auszublenden, dies ist unter anderem nützlich bei der Darstellung der initial approximierten Positionen der Personen.

go-Button

Schließlich kann die Simulation mit dem `go`-Button gestartet und pausiert werden, da hier die *forever*-Option aktiv ist. Ist diese deaktiviert, wird pro Betätigung des `go`-Buttons nur ein *Tick* durchgeführt.

2.1.2 Generelle Parameter

Der Parameter `input-file` erlaubt die Definition des Grundrisses der Simulationsumgebung. Während des Setups wird der Parameter zur Pfadauflösung für eine Bild-Datei verwendet. Diese wird mit dem Befehl `import-pcolors inputFile` auf die Simulationswelt gemappt.

`input-file` $\in \{$*Abstract.png*, *Abstract_static.png*, *Simple.png*, *Raumplan.png*$\}$

Der nächste generelle Parameter ist `orientation-algorithm`. Dieser dient der Auswahl eines Algorithmus zur Orientierung und Lokalisierung der Personen. Detaillierte Informationen sind im Kapitel 3 aufgeführt.

`orientation-algorithm` $\in \{$*Cellular automaton*, *Gradient localization*$\}$

Mit dem `graph-type`-Parameter hat der Nutzer die Wahl zwischen den Graphtypen, die in der Vorlesung vorgestellt wurden. Sofern *UDG* gewählt ist, wird der `person-detection-radius`-Parameter des folgenden Abschnitts für den Disk-Radius verwendet.

`graph-type` $\in \{Complete\ Graph, UDG, RNG, GG\}$

2.1.3 Personen–Parameter

`person-count` definiert die Anzahl der Personen in der Simulationsumgebung.

`person-count` $\in [1, 300]$

Mit dem `walk-probability`-Parameter wird die Wahrscheinlichkeit definiert, mit der Personen bei einem Tick einen Schritt machen. Bei 0% werden die Personen statisch an der gegenwärtigen Position fixiert. Es ist also möglich diesen Parameter während der Laufzeit zu verändern.

`walk-probability` $\in [0, 100]$

Der `walk-strategy`-Parameter erlaubt die Auswahl der *random walk*-Strategie (siehe Abschnitt 2.2.2).

`walk-strategy` $\in \{$*Complete random*, *Straight with collision detection*, *Straight with probability*$\}$

Analog zur `walk-probability` kann der Nutzer den `random-walk-probability`-Parameter zur Laufzeit anpassen und somit die Wahrscheinlichkeit der *random walk* Richtungsänderung bestimmen. Bei 100% wird jede Person nach jedem Tick eine Richtungsänderung vornehmen.

`random-walk-probability` $\in [0, 100]$

Der `person-detection-radius`-Parameter ist einer der entscheidendsten bei der Simulation. Hiermit wird der Radius definiert in dem eine Person eine Gefahrensituation wahrnehmen kann, sowie die Kommunikationsreichweite bei dem UDG-Graphen

bestimmt. Zudem wird damit der Abstand zu einem Notausgang bestimmt (siehe Kapitel 3.

`person-detection-radius` $\in [0, 700]$

2.1.4 Event–Parameter

Mit dem Parameter `event-count` wird die Anzahl der zu platzierenden Gefahrenevents festgelegt. Bei `event-count`$= 0$ wird es zu keiner Gefahrensituation kommen, sodass der random-walk getestet werden kann.

`event-count` $\in [0, 20]$

In der Aufgabenstellung wird ein zufälliges Auslösen von Gefahrensituationen gefordert, die beiden Parameter `min-countdown` und `max-countdown` bewerkstelligen dies. Jedes einzelne Gefahrenevent erhält zufällig einen initialen Countdown im Intervall [`min-countdown`, `max-countdown`]. Die obere bzw. untere Intervallgrenze der Parameter wird durch den jeweils anderen Parameter eingeschränkt.

`min-countdown` $\in [1, \textit{max-countdown}]$

`max-countdown` $\in [\textit{min-countdown}, 100]$

Der Parameter `gas-expansion-probability` legt für alle Gefahrenevents die Ausbreitungswahrscheinlichkeit und somit die Ausbreitungsgeschwindigkeit fest. Bei 0% wird nur genau ein Patch unter dem jeweiligen Gefahrenevent zu einer Bedrohung für die Personen. Personen können die Gefahrensituationen somit wahrnehmen, die Wahrscheinlichkeit das Personen sterben ist jedoch sehr gering.

`gas-expansion-probability` $\in [0, 100]$

2.1.5 Notausgang–Parameter

Der Parameter zur Einstellung der verfügbaren Notausgänge in der simulierten Welt, `number-of-exits` ist sehr bedeutsam bei der Lokalisierungsgenauigkeit und dem Fluchtverhalten der Personen.

`number-of-exits` $\in [1, 9]$

Zur dezentralen Orientierung der Personen und Schaffung eines optimalen Fluchtweges, wird auf den Zellulären Automaten zurückgegriffen und die Patches mit Informationen zur Signalstärke jedes Notausganges versehen.

Der Parameter `exit-signal-strength` bildet die maximale Signalstärke bzw. Reichweite der Notausgänge ab. Es ist möglich, dass nicht jedes Patch mit allen Signalinformationen versehen ist, da die Reichweite eines Notausganges zu gering war.

`exit-signal-strength` $\in [0, 1000]$

Die Aufgabenstellung fordert eine Limitierung Fluchtkapazitäten von Notausgängen. D.h. Notausgänge können maximal `exit-limit` Personen pro Tick evakuieren, sonst werden sie blockiert.

`exit-limit` $\in [1, 300]$

2.1.6 Multilateration localization–Parameter

Der Parameter `locate-iterations` bestimmt, wieviele Iterationen der Algorithmus durchführen soll. Je höher diese Zahl ist, desto genauer wird die lokalisierung mit dem Algorithmus, jedoch erfordert er dann auch mehr Rechenzeit. Dieser Parameter ist für die Evaluierung des Multilateration Algorithmus nötig, da man so bestimmen kann, ab wievielen Iterationen bereits vernünftige Ergebnisse herauskommen und ab wann es sich nicht mehr lohnt in Relation mit der Rechenzeit.

`locate-iterations` $\in [0, 100]$

Da der Algorithmus den *Hop Count* (vorgestellt in 3.1) benutzt um die Distanz zu den Bezugspunkten (in diesem Fall die Notausgänge) zu bestimmen, braucht der Algorithmus auch eine Abschätzung, wieviele Patches ein hop zu einer Person repräsentiert. Das heißt, eine Person mit einem hop count n zum Notausgang x hat eine geschätzte Distanz zu x von n * approx-dist. Für die Evaluation müssen verschiedene Werte ausprobiert werden, abhängig davon, wie der Baum aufgebaut wird und wieviele Personen in der Simulation leben.

`approx-dist` $\in [0, 200]$

2.2 Personen

Personen sind Agenten, die mit NetLogo als *breed* modelliert werden. Mit einem *breed* kann das Konzept der Kapselung aus der Objektorientierung realisiert werden. Im Kontext einer Person können lokale Variablen deklariert werden, die von den abstrakten Variablen des *breeds* vererbt werden.

Mit diesem Konzept wird ein Lebenszyklus mit verschiedenen Zuständen und Zustandsübergängen für die Personen modelliert.

2.2.1 Lebenszyklus

Der Lebenszyklus bestimmt das Verhalten der Personen. Die Zustände und die Zustandsübergänge werden in Abbildung 2.2 mittels eines Zustandsdiagramms beschrieben.

Nach der Platzierung in der simulierten Welt befinden sich alle Personen im Zustand `INIT`. In diesem Zustand wird ein *random walk* ausgeführt. Die verschiedenen Typen werden im Abschnitt 2.2.2 näher erläutert.

Nimmt eine Person in ihrem Sichtradius `person-detection-radius` ein Gefahrenevent war, geht die Person in den Zustand `EVENT_DETECTED` über. Hier versucht die Person andere Personen in der Umgebung zu warnen. Näheres ist dem Kommunikationsmodell im Abschnitt 2.2.3 zu entnehmen.

Nach dem Benachrichtigungsversuch wechselt die Person in den Zustand `FLEEING`, bei dem die Person vor der Gefahrensituation flüchtet und versucht einen Notausgang zu erreichen. Zur Flucht wird ein anderes Bewegungsmodell verwendet, welches im Abschnitt 2.2.2 beschrieben wird. Während der Flucht kann die Person weitere Gefahrensituation detektieren.

Erreicht die Person einen nicht blockierten Notausgang, ist sie gerettet und im Zustand `RESCUED`. Die Person wird aus der Simulationsumgebung entfernt.

Bei einer Berührung mit dem Giftgas stirbt eine Person jedoch immer. Sie ist dann im Zustand `DEAD`. Eine Interaktion ist mit ihr nicht mehr möglich.

2.2.2 Bewegungsmodell

Für die Personen existieren zwei unterschiedliche Bewegungsmodelle, zum einen für Personen im `INIT` Zustand und zum anderen für Personen im `FLEEING` Zustand. Beide Modelle werden folgend erklärt.

Initiales Bewegungsmodell

Für das initiale Bewegungsmodell kann der Nutzer zwischen drei Arten eines *random walks* wählen. Der rudimentäre *random walk* wird in Algorithmus 1 beschrieben. Dieser dient als Grundlage für die weiteren Arten.

Algorithmus 1 beschreibt die Wahl eines benachbarten Patches, welche als Wahl einer von acht neuen Richtungen interpretiert werden kann. Bei dem Algorithmus wird lediglich auf eine Wand-Kollision geprüft und solange eine Wand in der beabsichtigten Richtung steht, wird eine neue Richtung gesucht.

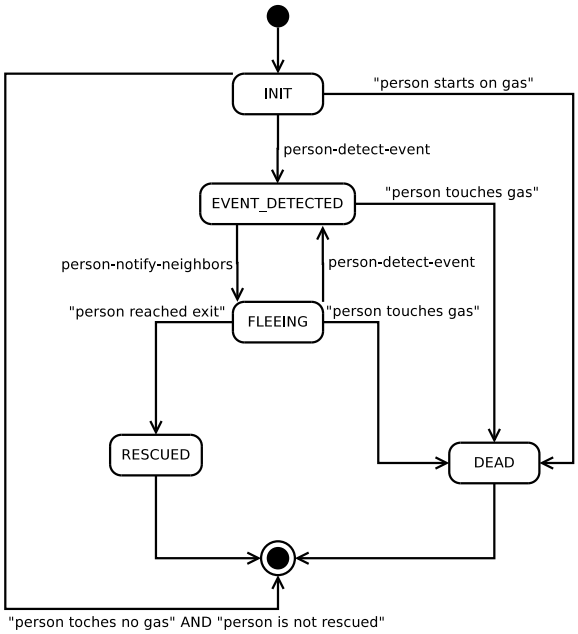

Abbildung 2.2: Zustandsdiagramm der Personen

Algorithmus 1 random-walk

nb ← one-of neighbors
while *[patch-state] of nb = WALL* **do**
  nb ← one-of neighbors
end while
face nb
forward 1

Eine weitere Strategie für den *random walk* ist `straight with collision detection`. Dabei speichert eine Person lokal die letzte Orientierung, das sogenannte `heading`, und bewegt sich in diese Richtung bis zu einer Wand-Detektion. An dieser Stelle wird Algorithmus 1 ausgeführt und die neue Orientierung gespeichert.

Bei der letzten Strategie `straight with probability` wird auch die letzte Orientierung lokal gespeichert, hier bestimmt jedoch der `random-walk-probability`-Parameter die Wahrscheinlichkeit der Ausführung von Algorithmus 1. Eine Wand-Detektion wird zusätzlich durchgeführt.

Bewegungsmodell bei der Flucht

Nachdem eine Person ein Gefahrenevent wahrgenommen hat, flieht sie zu dem nächsten verfügbaren Notausgang. Der optimale Fluchtweg wird über Informationen der Signal-Stärke von Notausgängen von den Personen lokal ermittelt. Abbildung 2.3 zeigt exemplarisch den Fluchtweg einer Person. Für die Algorithmik zur Wertevergabe auf den Patches sei auf Kapitel 3 verwiesen.

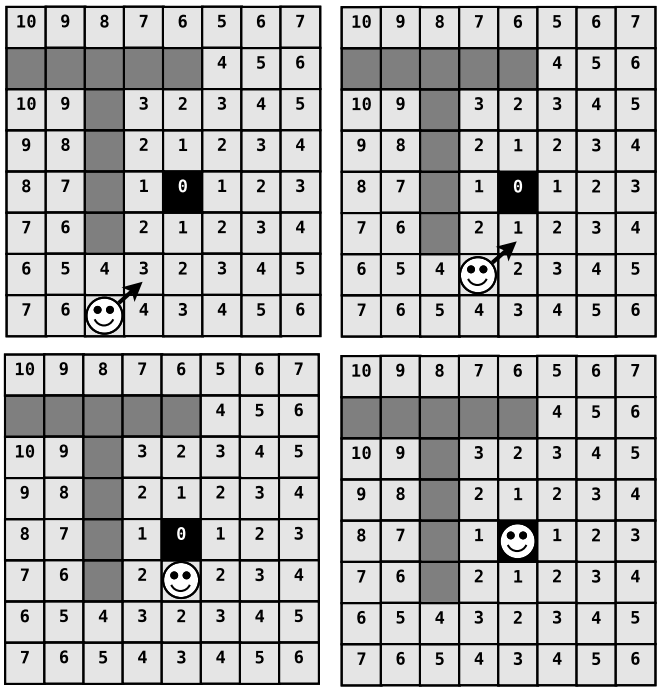

Abbildung 2.3: Fluchtwegermittlung

Der Algorithmus für die Fluchtwegbestimmung wird auf Grund der hohen Komplexität an dieser Stelle nicht komplett vorgestellt. Da zum Beispiel die Verfügbarkeit

von Notausgängen oder Ausnahmebehandlungen berücksichtigt werden müssen. Das Prinzip verdeutlicht Algorithmus 2. Bei dem die Person in der lokalen Umgebung nach dem Patch mit der geringsten Zahl bzw. der geringsten Dämpfung des Signals eines Notausgangs sucht.

Algorithmus 2 person-move-to-exit

```
nb ← -1
min-noise ← ∞
ask neighbors [                                  ; iterate all 8 neighbors
if min-noise > signal-noise then
  min-noise ← signal-noise                       ; define best signal and direction
  nb ← self
end if
]
face nb
forward 1
```

Die Information über die Verfügbarkeit eines Notausganges wird von diesem durch das Kommunikationsnetzwerk der Personen gebroadcastet. Eine Beschreibung dazu ist dem Abschnitt 2.3.2 Kommunikationsmodell der Notausgänge zu entnehmen.

2.2.3 Kommunikationsmodell

Für die Warnung anderer Personen wird das Kommunikationsmodell *Basic Flooding* aus der Vorlesung implementiert. Die identifizierende Meldung lautet hier `event-detected` und die beiden Zustände für das *Basic Flooding* werden mit `EVENT_DETECTED` und `FLEEING` ausgedrückt.

Algorithmus 3 zeigt, zur besseren Einordnung, die Einbettung des Kommunkationsprotokolls in den Lebenszyklus der Person. Es sei darauf hingewiesen, dass auf eine spontane Zustandsänderung einer Person, wie es in dem *Basic Flooding*-Protokoll angegeben ist, verzichtet wird. Die Detektion einer Gefahrensituation präzisiert diesen Zustandsübergang in diesem Fall präziser.

Das Fluten mit Meldungen ist jedoch ohne eine sinnvolle Netzwerktopologie nicht möglich. Mit dem Parameter `graph-type` hat der Nutzer die Möglichkeit den Graph-Typen des Kommunikationsnetzwerks anzupassen.

Allerdings wird in den meisten Fällen keine statische Knoten-Positionierung vorliegen[1]. Es wird daher nach jedem Tick der gesamte Kommunikationsgraph neu

[1]Eine statische Knoten-Positionierung wird mit *walk-probability* = 0 erzielt.

Algorithmus 3 Warnung vor Gefahrensituationen

State Trans. Sys.: ⟨{INIT, EVENT_DETECTED, FLEEING, RESCUED}, {INIT, DEAD}, {INIT, EVENT_DETECTED, DEAD}, {INIT, EVENT_DETECTED, FLEEING, DEAD}, {FLEEING, EVENT_DETECTED}⟩
Initialization: All notes in state INIT
Restrictions: Reliable communication; connected, bidirected communication graph $G = (V, E)$, neighborhoodfunction nbr: $V \rightarrow 2^V$
Local data:

INIT
Receiving(*event_detected*)
while *not event_detected* **do**
  random-walk
  create-graph(G) *; generate complete new graph*
end while
become EVENT_DETECTED
if touching-gas **then**
  become DEAD
end if

EVENT_DETECTED
broadcast(*event_detected*) *; broadcast event detection to linked neighbors*
become FLEEING
if touching-gas **then**
  become DEAD
end if

FLEEING
if msg = event-detected **then**
  broadcast(*event_detected*)*; forwarding event detection msg to linked neighbors*
end if
while *not person-reach-exit* **do**
  person-move-to-exit *; using orientation-algorithm*
  create-graph(G) *; generate complete new graph*
  if event_detected **then**
    become EVENT_DETECTED
  end if
  if touching-gas **then**
    become DEAD
  end if
end while
become RESCUED

RESCUED
create-graph(G) *; generate complete new graph without note*

DEAD
create-graph(G) *; generate complete new graph without note*

aufgebaut. Auf eine Ausnahmebehandlung für Personen ohne Positionsänderung wird verzichtet.

Zur visuellen Unterstützung erhalten Personen im Zustand `EVENT_DETECTED` das Label „!“. Der initiale Broadcast zur Warnung anderer Personen wird mit orange gefärbten Kanten und der Zustand `FLEEING` mit dem Label „*“ ausgedrückt.

2.3 Notausgänge

Notausgänge sind statische Agenten, für die eine eigene *breed* analog zu den Personen angelegt wurde. Die Notausgänge verfügen über einen Lebenszyklus und ein Kommunikationsmodell zur Übermittlung ihrer Zustandsinformationen an nahe Personen.

2.3.1 Lebenszyklus

Der Lebenszyklus eines Notausgangs ist in drei Zustände unterteilt. Im Zustand `INIT` befinden sich alle Notausgänge nach der Platzierung auf der Welt. Während des Zustandsübergangs zu `NEGOTIABLE` wird die Logik für den gewählten Orientierungsalgorithmus ausgeführt. In der Regel wird die Welt mit Signalinformationen ausgehend von jedem Notausgang geflutet. Der Algorithmus wird in Kapitel 3 vorgestellt. Die Ausführung dauert entsprechend der (Patch-)Auflösung lange.

Ist ein Notausgang im Zustand `NEGOTIABLE`, bestehen zwei Möglichkeiten in den Zustand `BLOCKED` zu gelangen. Temporär blockiert ist ein Notausgang, wenn zu viele Personen gleichzeitig versuchen das Gebäude zu verlassen. Der Parameter `exit-limit` definiert die Obergrenze. Nach einem Tick wird der Notausgang wieder zurückgesetzt. Erreicht das Giftgas bzw. die Gefahrensituation einen Notausgang, so wird dieser permanent in den Zustand `BLOCKED` über.

2.3.2 Kommunikationsmodell

Die Notausgänge befinden sich im gleichen Kommunikationsnetzwerk, wie die Personen. Eine Unterscheidung zu anderen Netzwerken bzw. Links wird über Typ-Definitionen und visuell über den Shape durchgeführt.

Für die Verbreitung der Passierbarkeit wird das *Basic Flooding* implementiert. Algorithmus 4 zeigt das eingebettete Protokoll innerhalb des Lebenszyklus von Notausgängen.

Algorithmus 4 Passierbarkeitsmeldungen der Notausgänge

State Trans. Sys.: $\langle$\{INIT, NEGOTIABLE, BLOCKED\}, \{NEGOTIABLE, BLOCKED\}, \{BLOCKED, NEGOTIABLE\}$\rangle$
Initialization: All notes in state INIT
Restrictions: Reliable communication; connected, bidirected communication graph $G = (V, E)$, neighborhoodfunction nbr: $V \to 2^V$
Local data: current-persons, exit-limit

INIT
... *; init-orientation-algorithm*
become NEGOTIABLE

NEGOTIABLE
negotiable ← true
while *negotiable* **do**
  broadcast(*exit-negotiable*) *; continious broadcast*
  if *current-persons* > *exit-limit* **then**
    negotiable ← false
  end if
  if *gas-reached-exit* **then**
    negotiable ← false
  end if
end while
become BLOCKED

BLOCKED
negotiable ← false
while *notnegotiable* **do**
  broadcast(*exit-blocked*) *; continious broadcast*
  if *current-persons* <= *exit-limit* **then**
    negotiable ← true
  end if
end while
become NEGOTIABLE

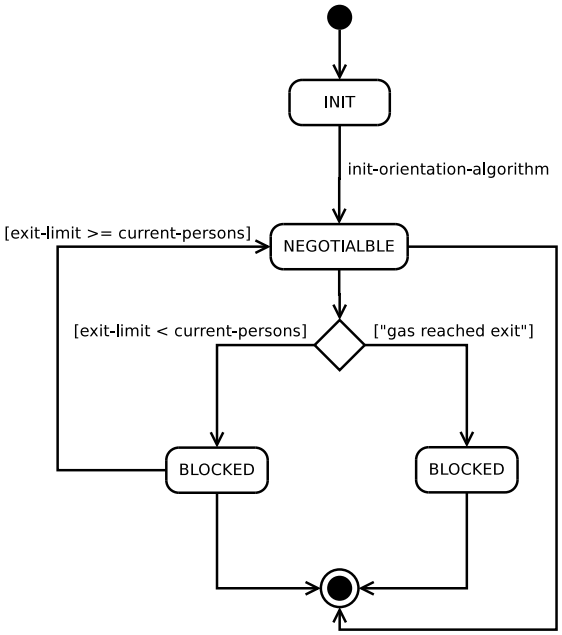

Abbildung 2.4: Zustandsdiagramm der Notausgänge

Die Notausgänge broadcasten bei Statuswechsel eine positive oder negative Passierbarkeitsmeldung und ihren Identifier im Netzwerk. Personen in Reichweite empfangen die Meldung `exit-blocked` oder `exit-negotiable`, speichern die Information lokal in einem binären Array ab und broadcasten die Meldung an die benachbarten Personen. Bei n = 3 Notausgängen verfügt jede Person über ein n-äres Array mit Nullen oder Einsen. Null steht für blockiert, Eins für passierbar.

2.4 Gefahrensituationen

Eine Gefahrensituation liegt vor, wenn nur ein Gefahrenevent aktiviert wurde. Die Gefahrenevents werden als *breed* modelliert.

2.4.1 Lebenszyklus

Die Zustandsübergänge von Gefahrenevents verlaufen sequentiell. Alle Events beginnen nach der Platzierung im Zustand `INIT` und enden im Zustand `DONE`. Abbildung 2.5 zeigt ebenfalls die beiden weiteren Zustände `COUNTDOWN` und `GASSING`.

Mit dem Start der Simulation gehen alle Events in den Zustand `COUNTDOWN` über, in

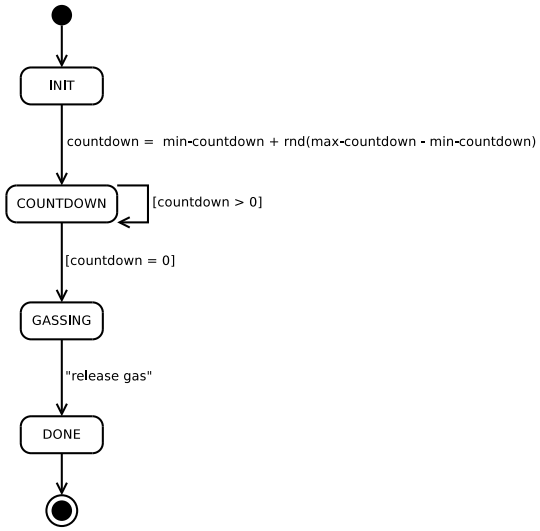

Abbildung 2.5: Zustandsdiagramm der Gefahrenevents

der lokal vorgehaltene zufällige Countdown pro Tick herunter gezählt wird. Steht der Countdown bei Null, geht das Gefahrenevent in den Zustand `GASSING` über.

In diesem Zustand wird die Freisetzung des Gases initiiert. Die Ausbreitung des Gases ähnelt dem Fluten mit Signal-Informationen auf Kapitel 3.

2.5 Patches

Die Patches können in NetLogo nicht als *breed* modelliert werden. Die Funktionsaufrufe aus dem Kontext eines Patches sind daher problematisch. Es können keine Kontrollstrukturen oder ordentliche Zustandsübergänge zentral erstellt werden. Es wird daher von impliziten Zuständen von Patches die Rede sein.

2.5.1 Implizite Zustände

Mit der Initialisierung aller der Patches nach dem Einbinden eines Grundrisses passiert die Zustandsfestlegung anhand der Farbe eines Patches. Für diesen Schritt kann generell zwischen zwei Zuständen unterschieden werden. `NONE` wird als leerer Raum angenommen, in dem sich Signal mit geringer Dämpfung ausbreiten können, Personen sich bewegen können ebenso wie das Giftgas. Der leere Raum wird im Grundriss über die Farbe `white` ausgedrückt.

Patches mit der Farbe `black` oder alle anderen Farben werden als Wände und Materialien mit hoher Dämpfung interpretiert. Die Patches gehen in den Zustand `WALL` über. Dieser Zustand ist statisch.

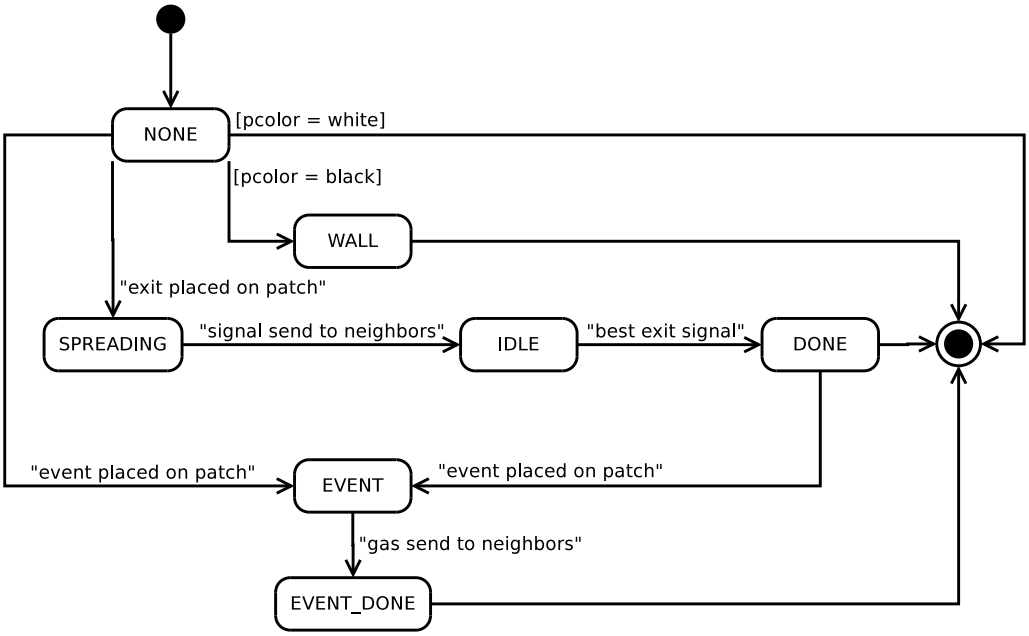

Abbildung 2.6: Zustandsdiagramm der Patches

Auf Patches im Zustand `NONE` werden die Personen, die Notausgänge und die Gefahrenevents platziert. Außerdem werden nur diese Patches mit Signal-Informationen geflutet. Ist dies der Fall, so nehmen die Patches die Zustände `SPREADING`, `IDLE` und `DONE` an.

Patches im Zustand `SPREADING` sind erstmalig von einem Notausgang mit Signalinformationen versehen worden. Patches in diesem Zustand verteilen die Signalstärke bzw. die Dämpfung an ihre benachbarten Patches, bei jedem Schritt wird die Dämpfung inkrementiert. Die Patches direkt unter einem Notausgang sind dabei die Ausgangspunkte für das Fluten.

Sobald ein Patch die Signal-Informationen von allen Notausgängen lokal gespeichert hat, wechselt es in den Zustand `IDLE`. Hier wird auf ein besseres Signal eines jeden Notausgangs gewartet. Dies ist der Fall bei anderen Lauf- bzw. Ausbreitungswegen der Signalen insbesondere bei Grundrissen mit vielen punktuellen Wandfragmenten der Fall. Wird ein besseres Signal festgestellt, wird es verständlicherweise an die benachbarten Patches gemeldet.

Haben alle Patches in der Nachbarschaft einen optimalen Signalwert, geht dieses

Patch in den Zustand `DONE` über. Allerdings wird dieser Idealzustand nicht für alle Patches erreicht, insbesondere bei einem niedrigen `exit-signal-strength`-Parameter. Eine Überlappung von zwei bis drei Signalen von Notausgängen ist in fast allen Fällen ausreichend für eine intelligente Flucht. Ist der nächstgelegene Notausgang blockiert, wird der zweitbeste Notausgang von der Person zur Evakuierung genutzt. Mit der Fortbewegung erhält sie permanent neue Informationen zu nahe gelegene oder blockierte Notausgänge.

Nach Abschluss der Signal-Flutung befinden sich die Patches entweder im Zustand `NONE`, sofern die Signalreichweite aller Notausgänge zu gering ist, oder in `DONE` mit Signalinformationen.

Analog zum Fluten mit Signal-Informationen initiieren Gefahrenevents direkt unter sich ein Patch im Zustand `EVENT`. Diese Gasfreisetzung bereitet sich genau wie die Signale aus, es wird jedoch nur der Zustand der Patches angepasst. Außerdem ist die Gasausbreitung an Nachbarn als einmalig initial und darauf folgend als kontinuierlich anzusehen. Patches, die die Event-Information an ihre Nachbarn weitergegeben haben, wechseln in den Zustand `EVENT_DONE`. Die Patches werden programmatisch bei der weiteren Ausbreitung ausgeschlossen. Es ist eine Laufzeitreduzierung festzustellen.

Personen detektieren Patches im Zustand `EVENT` und `EVENT_DONE` als Gefahrensituation.

2.6 Ressourcen der Simulationsumgebung

Die Simulationsumgebung wird über die Datei `Evakuierung.nlogo` gestartet. Der Programmcode ist nach Funktion und *Breed*-Klasse unterteilt.

Gefahrensituationen

`event.nls`
`event-gassing.nls`

Die Quellcode-Datei `event.nls` beinhaltet den Lebenszyklus der Gefahrenevents und deren Zustandsübergangsprotokoll. Mittels `event-gassing.nls` wird die Ausbreitung des Giftgases implementiert, die größtenteils den Patch-Lebenszyklus manipuliert.

Lokalisierung

`locate.nls`

In dieser Datei wird der Algorithmus zur Lokalisierung aus dem Paper [2] implementiert.

Notausgänge

`exit.nls`
`exit-cellular-automaton.nls`
`exit-gsn.nls`

Die Quellcode-Datei `event.nls` steuert den Setup und den Lebenszyklus der Notausgänge. Mittels `exit-cellular-automaton.nls` wird der Orientierungsalgorithmus auf Basis des zellulären Automats realisiert. Letztlich definiert die Datei `exit-gsn.nls` das Kommunikationsmodell zur Übermittlung der Statusinformationen.

Personen

`person.nls`
`person-gsn.nls`
`person-linking.nls`

`person.nls` regelt den Setup der Personen und deren Lebenszyklus. `person-gsn.nls` umfasst den Quellcode für die Kommunikation zwischen Personen und mit der Datei `person-linking.nls` wird der Graph zwischen Personen erstellt.

Simulationswelt

`patch.nls`

Hier wird der Quellcode für den Lebenszyklus der *Patches* definiert.

3 Algorithmik

3.1 Multilateration Lokalisierung

Der Algorithmus startet beim Setup der Simulation mit der Bestimmung der Hop Counts von allen Personen zu allen Ausgängen. Der *Hop Count* repräsentiert im Netzwerk die Anzahl der Sprünge, die benötigt wird, um einen Knoten zu erreichen. Sei beispielsweise ein Notausgang mit einer Person verbunden, so hat sie einen Hop Count von 1 vom Notausgang aus. Ist diese Person mit einer anderen Person verbunden, die wiederum *nicht* mit dem Notausgang verbunden ist, so hat sie einen Hop Count von 2.

Theorie Nachdem die geschätzten Distanzen zu den Orientierungspunkten (Notausgängen) zu jeder Person berechnet sind, ist es möglich, eine Abschätzung der Position dieser Person zu machen. Dafür benötigt man mindestens 3 Punkte, es werden jedoch wesentlich mehr Punkte benötigt, um eine bessere Approximation zu der echten Position zu erreichen (vergleiche [2] ab Seite 217, Kapitel Analysis).

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3.1}$$

In der Formel 3.1 bestimmen wir die Formel, die die Distanz zwischen einem Orientierungspunkt j und einem beliebigen Punkt i beschreibt. Das benutzen wir, um die tatsächliche Distanz zu der Person mit der Distanz zum geschätzten Punkt zu vergleichen.

$$E_j = \sum_{i=1}^{n} (d_{ji} - \hat{d}_{ji})^2 \tag{3.2}$$

In der Formel 3.2 bestimmt die Summe des quadrierten Fehler, d.h. die Fehleinschätzung unserer geschätzten Distanz zu der tatsächlichen Distanz. Dabei ist d die tatsächliche Position und $\hat{d}$ die geschätzte Position. Nun ist also d die unbekannte und wir wollen diesen Fehler minimieren.

$$\frac{\partial E_j}{\partial x_i} = \sum_{i=1}^{n}(x_j - x_i)\left(1 - \frac{d_{ji}}{\hat{d}_{ji}}\right) \text{ and } \frac{\partial E_j}{\partial y_i} = \sum_{i=1}^{n}(y_j - y_i)\left(1 - \frac{d_{ji}}{\hat{d}_{ji}}\right) \tag{3.3}$$

Um den Fehler zu minimieren bilden wir die partiellen Ableitungen in 3.3 und versuchen so, nach Methoden der Analysis, das Minimum zu bestimmen. Wir starten also mit einer geschätzten Position $\hat{d}$ und verschieben langsam die Punkte um α und minimieren damit den Fehler.

Abbildung 3.1: Visualisierung der geschätzten Positionen

Implementierung In der Implementierung wählen wir als Punkt $\hat{d}$ den Ausgang, der dem Charakter laut hop count am nächsten steht, da jede Person die Position aller Notausgänge im Netzwerk kennt. Bei der Implementierung orientiert sich der Algorithmus an den Referenzcode der zur Verfügung gestellt wurde. Nachdem der Hop Count beim Setup für jede Person bestimmt wurde, kann der Algorithmus für jede Person ausgeführt werden. Wieviele Iterationen vorgenommen werden, ist ein einstellbarer Parameter. Nachdem eine Position für eine Person geschätzt wurde, wird sie als rote Person in die Simulationsumgebung integriert und mit der richtigen Person mit einem roten Link verbunden, wie auf der Abbildung 3.1 zu sehen ist.

3.2 Zellulärer Automat

Die mobilen Geräte bestimmen über die *Gradienten Lokalisierung* die approximierte Position der Personen. Daraus wird die Entfernung zu den Notausgängen geschätzt. Im Falle einer Gefahrensituation flüchten die Personen zu dem nächst gelegenen und verfügbaren Notausgang.

Für die Orientierung während der Flucht steuert die Umgebung (Zelluläre Automaten) die Bewegung der Person.

Die zellulären Automaten dienen zur Modellierung räumlich diskreter dynamischer Systeme. In diesem Fall wird die zweidimensionale Simulationsumgebung als Zellularraum Z und die Patches als Zellen $z \in Z$ betrachtet. Die Patches bilden ein orthogonales Gitter.

Für die endliche Nachbarschaft der Zellen ergibt sich:

$$N_{Moore}(z) = \begin{cases} |Z_{Neighbor}| = 3, & \text{falls } Edge(z) = true, \\ |Z_{Neighbor}| = 5, & \text{falls } Border(z) = true, \\ |Z_{Neighbor}| = 8, & \text{sonst.} \end{cases}$$

$$N_{Neumann}(z) = \begin{cases} |Z_{Neighbor}| = 2, & \text{falls } Edge(z) = true, \\ |Z_{Neighbor}| = 3, & \text{falls } Border(z) = true, \\ |Z_{Neighbor}| = 4, & \text{sonst.} \end{cases}$$

Abbildung 3.2 visualisiert die beiden Nachbarschaftsbeziehungen, Moore-Nachbarschaft $N_{Moore}(z)$ oder Von-Neumann-Nachbarschaft $N_{Neumann}(z)$, für die drei Zelltypen. Zelle z_1 ist an der oberen Ecke des Raumes, Zelle z_2 am Rand und Zelle z_3 innerhalb des Zellularraums.

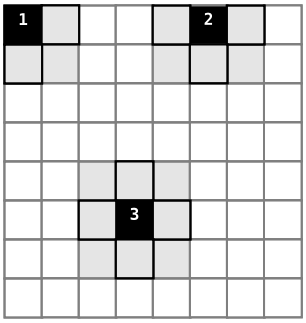

Abbildung 3.2: Nachbarschaftsbeziehungen der Zellen

Neben der Menge an benachbarten Zellen besitzt jede Zelle einen diskreten Zustand $q \in Q$ aus der Zustandsmenge Q, die bereits in Abschnitt 2.5.1 angesprochen wurde.

$$Q = \{NONE, SPREADING, IDLE, DONE\}$$

Zusätzlich besitzt jede Zelle lokale Zustandsübergangsregeln $\delta: Q^N \rightarrow Q$, die in Abhängigkeit mit den Zuständen und Werte aller Nachbarn zum Zeitpunkt t deren neuen Zustand bestimmt. Für jeden diskreten Zeitschritt $t + 1$ werden die Zustandsübergangsregeln für alle Zellen angewendet.

Besonders ist hier, dass sowohl der Zustand einer Zelle überführt wird, als auch die Signalausbreitung der Notausgänge simuliert wird. Jede benachbarte Zelle $Z_{Neighbor} \subset Z \backslash \{z\}$ erhält den inkrementierten Signal-Dämpfungswert des Vorgängers $z \in Z$. Nachdem die dynamische Signalausbreitung abgeschlossen ist, werden die zellulären Automaten als terminiert, d.h. statisch angesehen. Zustands- und Wertänderungen sind nicht mehr möglich. Die Signalausbreitung ist beendet, sobald

die maximale Dämpfung erreicht wurde.

Abbildung 3.3 zeigt die Zeitschritte t_1 bis t_9 bei Anwendung der Von-Neumann-Nachbarschaftsbeziehung. Die schwarze Zelle markiert die Position eines Notausgangs. Zum Zeitpunkt t_0 ist wird diese Zelle mit dem Signal-Dämpfungswert = 0 und dem Zustand `SPREADING` (spontan) initialisiert.

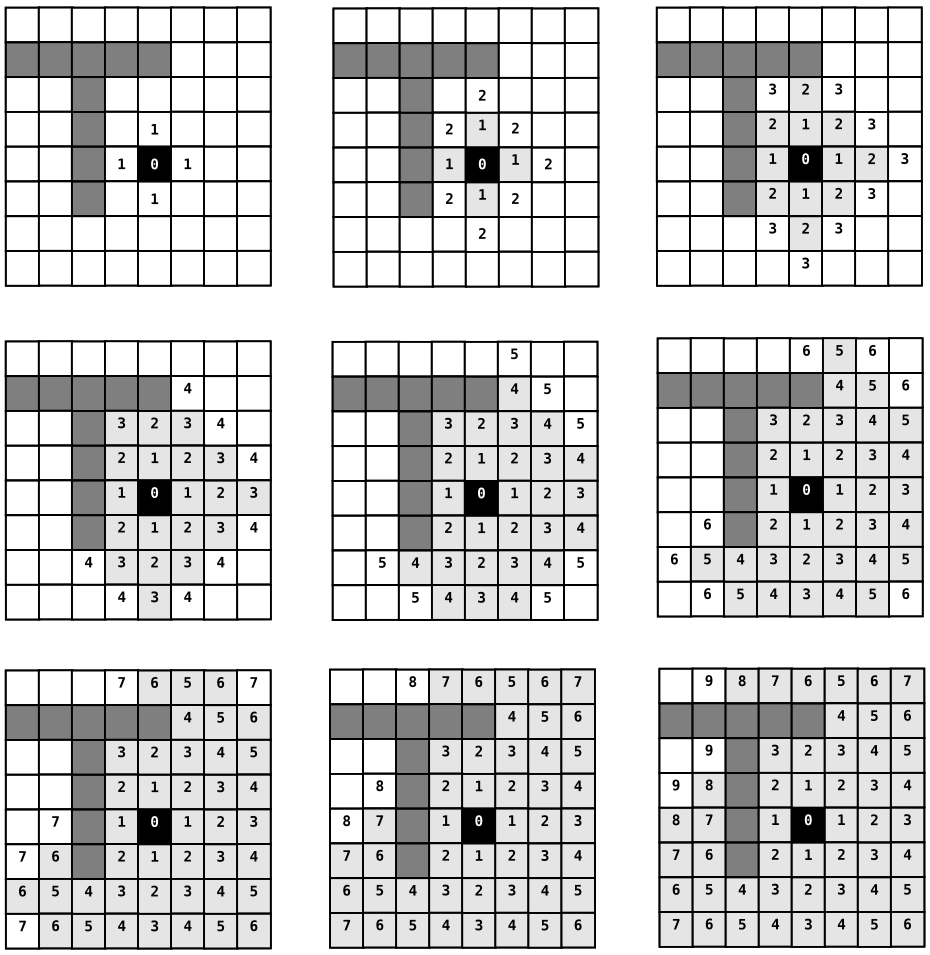

Abbildung 3.3: Signalausbreitung eines Notausganges (Zelluläre Automaten)

Die Signalausbreitung unterliegt dem lokalen Verhalten der zellulären Automaten durch die entsprechenden Zustandsübergangsregeln. Bei $i \geq 3$ Notausgängen sind die Zustandsübergangsregeln entsprechend komplex[1] und werden an dieser Stelle nicht detailliert beschrieben.

[1] Nach der Klassifizierung durch [4] können die zellulären Automaten der Klasse 4 zugeordnet werden.

3.2.1 Orientierung der Personen bei der Flucht

Nachdem die Signalausbreitung abgeschlossen ist, orientiert sich eine Person bei der Flucht anhand der lokalen Signal-Dämpfungswerten der aktuellen Zelle und der benachbarten Zellen. Das Bewegungsmodell wurde bereits in Abschnitt 2.2.2 angeschnitten. Abbildung 2.3 dient daher als visuelle Unterstützung der Fluchtwegbestimmung.

Zum Zeitpunkt t_{-1} detektiert die Person eine Gefahrensituation. Die Person hat, über die *Gradienten Lokalisierung* und der Kommunikation der Notausgänge, den Notausgang auf $z_{exit} \in Z$ als nächstes verfügbares Evakuierungsziel bestimmt.

Die Person sei bei t_0 auf Zelle $z_0 \in Z$ positioniert. Die Nachbarschaft sei mit $N_{Moore}(z_0) = Z_{z_0}$ gegeben. Anders als bei der Signalausbreitung bestimmt die Person den Fluchtweg über die Moore-Nachbarschaft. Der Vorteil liegt in der Möglichkeit diagonale Bewegungen auszuführen.

Die Ziel-Zelle $z_i \in Z$ für den nächsten Zeitschritt t_i bestimmt die Person mittels der Funktion $N_i(Z_{z_{i-1}})$. Die Funktion $SN(z)$ liefert den Signal-Dämpfungswert einer Zelle.

$$N_i(Z_{z_{i-1}}) := min(SN(z_{0_u})) = z_i,\ u = 1, ..., \mid Z_{z_i} \mid$$

Entsprechend der Abbildung 2.3 bestimmt die Person t_1, t_2 und t_3 folgende Ziel-Zellen:

$$(t_1)\ z_1 = N_1(Z_{z_0}) = min(5, 4, 3, 4, \infty, \infty, \infty, 6).\ SN(z_1) = 3$$

$$(t_2)\ z_2 = N_2(Z_{z_1}) = min(\infty, 2, 1, 2, 3, 4, 5, 4).\ SN(z_2) = 1$$

$$(t_3)\ z_3 = N_3(Z_{z_2}) = min(1, 0, 1, 2, 3, 2, 3, 2).\ SN(z_3) = 0$$

Erreicht eine Person eine Zelle mit dem Signal-Dämpfungswert = 0, versucht sie sich zu retten. Ist das Limit des Notausgangs bereits erreicht, sodass dieser blockiert, sucht sich die Person den nächstgelegenen verfügbaren Notausgang. Dafür ist eine Überlagerung der Signale von Notausgängen notwendig. Hinzu kommt die benötigte Verbindung zu einem verfügbaren Notausgang oder einer weiteren Person mit dem Wissen über einen verfügbaren Notausgang. Diese beiden Kriterien müssen erfüllt sein, sodass der Algorithmus zur Fluchtwegbestimmung funktioniert.

4 Evaluation

4.1 Lokalisierung

Um die beste Lokalisierung mit dem Algorithmus zu erreichen, spielen die Parameter die größte Rolle. In den folgenden Tabellen und Analysen wird deutlich, wie groß die Unterschiede zwischen den verschiedenen Ausführungen sein kann, wenn man einen Parameter verändert. Zwischen den Lokalisierungen in einer Tabelle werden die Personen nicht neu generiert, die Werte in einer Tabelle sind also vergleichbar. Zwischen verschiedenen Tabellen werden die Personen jedoch neu generiert, also kann es zu generellen Abweichungen kommen, da die Verteilung der Personen auch eine Rolle in der Genauigkeit spielt.

Beschreibung der Parameter und Tabellenspalten

number-of-exits (Tabellenspalte *#Exits*)
: Anzahl der Ausgänge und Sensorpunkte für den Algorithmus. Der Algorithmus soll mit mehr Sensorpunkten besser approxmieren, ab 3 ist eine realistische Berechnung überhaupt erst möglich.

personCount (Tabellenspalte *#Person*)
: Anzahl der Personen in der Simulation. Eine Mindestanzahl ist nötig, um das Kommunikationsnetz aufzubauen. Mehr Personen bedeutet eine genauere Abschätzung der Distanzen zu den Sensorpunkten, da der Hop Counts zuverlässiger ist. Das ist durch die Verteilung der Personen gegeben, bei sehr vielen Personen ist die Chance höher, das die Hop Counts *normaler* verteilt sind.

detection-radius (Tabellenspalte *detRad*)
: Radius mit dem der UDG bestimmt wird. Das Attribut ist auch für den Hop Count verantwortlich, wenn man ihn höher wählt, dann ist der Hop Count für weiterstehende Personen kleiner, da der Kommunikationsradius höher ist. Ein kleinerer Wert könnte dazu führen, dass das Netz nicht vollständig alle Notausgänge und Personen verbindet, zu groß führt zu einem ungenauen Hop Count.

locate-iterations (Tabellenspalte *#iter*)
Wieviele Iterationen der Algorithmus durchläuft. Mehr Iterationen bedeutet höhere Genauigkeit, aber auch höherer Rechenaufwand.

approx-dist (Tabellenspalte *dist*)
Die Distanz die der Algorithmus zu jedem Hop Count annimmt. Die Zahl ist abhängig von der Personendichte und dem detection-radius.

average-estimated-distance (Tabellenspalte *avgDist*)
Durchschnittlicher Abstand aller geschätzten Positionen zu der tatsächlichen Position.

min-estimated-distance (Tabellenspalte *minDist*)
Kleinste Distanz von einer geschätzten Position zu einer genauen Position, d.h. die am besten geschätzte Position.

max-estimated-distance (Tabellenspalte *maxDist*)
Größte Distanz von einer geschätzten Position zu einer genauen Position, d.h. die am schlechtesten geschätzte Position.

No	#Exits	#Person	detRad	#iter	dist	avgDist	minDist	maxDist
1	3	100	50	5	15	52,13	10,22	126,51
2	3	100	50	5	30	35,23	3,33	115,85
3	3	100	50	5	35	38,65	3,24	121,78
4	3	100	50	10	15	52,87	10,56	126,47
5	3	100	50	10	30	34,44	4,69	105,88
6	3	100	50	10	35	28,18	1,84	109,89
7	3	100	50	20	15	52,49	10,96	126,95
8	3	100	50	20	30	33,75	4,44	90,54
9	3	100	50	20	35	22,95	1,81	82,22
10	3	100	50	30	15	52,82	10,91	126,36
11	3	100	50	30	30	33,21	4,45	80,27
12	3	100	50	30	35	20,1	1,17	66,19

Tabelle 4.1: Evaluierung für raumplan.png mit 3 Exits und 100 Personen

An der Tabelle 4.1 kann man bei den Anzahl der Iterationen sehen, wie die Genauigkeit des Algorithmus mit der Anzahl der Iterationen steigt. Zwischen 5 und 20 Iterationen ist eine Verbesserung von 50% zu sehen, wenn die approx-dist dementsprechend gewählt wurde. Von 20 auf 30 Iterationen ist die Verbesserung in Relation mit dem Rechenaufwand wohl irrelevant. In den zukünfiigen Experimenten sei die Anzahl der Iterationen stets 20.

No	#Exits	#Person	detRad	#iter	dist	avgDist	minDist	maxDist
13	3	150	30	20	15	41,15	2,57	88,11
14	3	150	30	20	17	35,55	4,45	72,51
15	3	150	30	20	19	27,51	4,23	52,65
16	3	150	30	20	20	23,46	2,37	58,46

Tabelle 4.2: Evaluierung für raumplan.png mit 3 Exits und 150 Personen

An der Tabelle 4.2 wird der Effekt von verschiedenen approx-dist Werten veranschaulicht. Man findet eine gute Approximation bei $\frac{2}{3}$ vom Detection Radius. Wenn ie approx-dist noch höher gewählt wird, fallen zuviele geschätzte Position außerhalb der Simulationsumgebung an und das Ergebnis wird verfälscht. Dabei muss noch beachtet werden, dass der detection-radius gesenkt werden konnte, da nun mehr Personen vorhanden sind, die im Kommunikationsnetz den Hop Count genauer weiterreichen und repräsentieren können.

No	#Exits	#Person	detRad	#iter	dist	avgDist	minDist	maxDist
17	3	200	25	20	15	21,56	1,25	62,25
18	3	200	25	20	17	14,59	1,64	36,32
19	3	200	25	20	19	22,14	2,31	66,81

Tabelle 4.3: Evaluierung für raumplan.png mit 3 Exits und 200 Personen

Bei der Tabelle 4.3 ist verdeutlicht, dass bei einer größeren Anzahl von Personen eine höhere Dichte entsteht und durch einen kleineren detection-radius zusammen mit einem passenden approx-dist wesentlich genauere Positionen gefunden werden können. Hier ist auch erneut zu sehen, dass $\frac{2}{3}$ vom Detection Radius wohl ein sinnvoller Wert ist, gegeben einem sinnvollen detection-radius, der nicht viel zu hoch ist, für die vorliegende Dichte der Personen. In den folgenden Experimenten bleiben wir bei 200 Personen mit einem detection-radius von 25.

No	#Exits	#Person	detRad	#iter	dist	avgDist	minDist	maxDist
20	6	200	25	20	15	21,72	0,65	48,62
21	6	200	25	20	17	12,78	1,14	35,02
22	6	200	25	20	19	10,17	0,76	57,57

Tabelle 4.4: Evaluierung für raumplan.png mit 6 Exits und 200 Personen

In Tabelle 4.4 wird deutlich, wie wichtig es ist viele gleichverteilte Orientierungspunkte zu haben für den Algorithmus. Die Genauigkeit ist deutlich höher im Mittel, insbesondere der kleinste und größte Abstand ist wesentlich besser.

No	#Exits	#Person	detRad	#iter	dist	avgDist	minDist	maxDist
23	9	200	25	20	15	18,35	1,64	42,25
24	9	200	25	20	17	11,08	0,78	28,35
25	9	200	25	20	19	18,03	1,25	41,47

Tabelle 4.5: Evaluierung für raumplan.png mit 9 Exits und 200 Personen

Bei Tabelle 4.5 mit 9 Exits ist noch eine Verbesserung zu sehen, inbesondere wieder in der min und max distance. Bei mehr Orientierungspunkten, einem dichteren Netzwerk und mehr Personen wäre eine Genauigkeit auf wenige Patches genau problemlos möglich.

4.2 Evakuierungsdauer

Anzahl Notausgänge
20 Durchläufe
Zeit bis erste Person evakuiert
Zeit bis letzte Person evakuiert

4.3 Effizienz

4.4 Fazit

4.5 Ausblick

Alternativer Orientierungsalgorithmus

Als alternativer Orientierungsalgorithmus zur lokalen Fluchtwegfindung kann der *Bug-2-Algorithmus* dienen. Dieser stammt aus der Fahrzeugführung autonomer mobiler Roboter und ist ein reaktives Verfahren. Dabei wird eine Wand-Detektion und -Verfolgung durchgeführt. Zusätzlich ist die Orientierung der Person lokal vorzuhalten. Das lokale Speichern des Headings einer Person widerspricht hier nicht den Anforderungen.

Bei dem Bug-2-Algorithmus wird eine sogenannte *m-Linie* zwischen Person und dem Notausgang gebildet. Dazu wird die approximierte Position der Person genutzt. Die m-Linie kann als Luftlinie zwischen den beiden Punkten betrachtet werden.

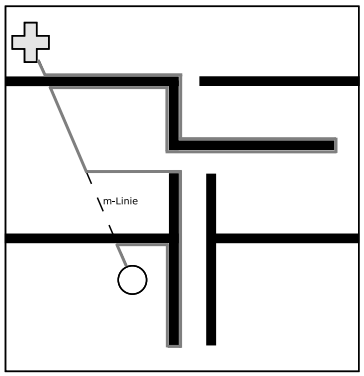

Abbildung 4.1: Veranschaulichung des Bug-2-Algorithmus

Abbildung 4.1 zeigt schematisch einen Grundriss mit einer Person bzw. einem Agenten und einem Notausgang. Die *m-Linie* ist gestrichelt dargestellt, die Wände schwarz.

Nach dem Auslösen der Flucht bewegt sich die Person entlang der m-Linie, bis eine Wand detektiert wird. Eine solche Detektion ist bereits bei dem *random walk* implementiert worden. Es wird von der Person nun ein sogenanntes *wall following* durchgeführt, bis die m-Linie näher am Notausgang passiert wird. Für diesen Schritt kann die initiale m-Linie lokal gespeichert werden oder falls möglich mit der approximierten Position eine neue m-Linie berechnet werden. Die Person bewegt sich erneut entlang der m-Linie.

Algorithmus 5 Bug-2-Algorithmus

```
last-heading ← GET-HEADING()
exit-position ← GET-NEAREST-EXIT()
m-line ← LINE(exit-position, approx-position)
while not rescued do
  if not wall-in-front then
    SET-HEADING(CALCULATE-HEADING(m-line))
    FORWARD 1
  else
    follow-wall ← true                                  ; wall detection
  end if
  while follow-wall do
    if on-m-line then
      follow-wall ← false                               ; abort wall following
    else
      last-heading ← FOLLOW-WALL(last-heading)          ; wall following
    end if
  end while
end while
```

Literaturverzeichnis

[1] Isaac Amundson and Xenofon D. Koutsoukos. *A Survey on Localization for Mobile Wireless Sensor Networks.* Department of Electrical Engineering and Computer Science, Vanderbilt University.

[2] Jonathan Bachrach, Radhika Nagpal, Michael Salib and Howard Shrobe. *Experimental Results for and Theoretical Analysis of a Self-Organizing Global Coordinate System for Ad Hoc Sensor Networks.* Telecommunication Systems, page 213–233. 2004.

[3] Uri Wilensky. *Netlogo.* Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. 1999. `http://ccl.northwestern.edu/netlogo/`, Stand: 26.01.2014.

[4] Stephen Wolfram. *A New Kind of Science.* Wolfram Media. 2002.