平成22年度　特別研究報告書

フロアフィールドモデルを用いた
ナビ情報の有効性検証

龍谷大学　理工学部　情報メディア学科

学籍番号 **T060607**　能見将裕

指導教員　三好　力　教授

内容梗概

セルオートマトン法は計算コストが小さいながらも人間の動きの特性を再現できることで知られており，本研究もセルオートマトン法を用い，その中でも特にフロアフィールドモデルと呼ばれるものを取り上げてシミュレーションを行う．　人間が地下街等の施設から避難行動を行う際に，移動先を決定するためには，出口までの最短距離と避難誘導灯という要素が重要となって来る．前者は最短距離で逃げ出そうとする人間の当たり前の心理を表しており，後者は地下街等の出口の位置が分からないような状況や迷ってしまった際などの移動指針としてとても大切なものである．

本研究では人が災害時など、避難を行う際に出口まで最短距離で移動しようとする傾向と，避難の際にその手助けとなる避難誘導灯という要素に注目し，簡易な袋小路が存在する部屋を例に取り，既存の手法を用いたモデルとそのモデルを参考に改変を行ったモデルそれぞれで誘導灯の設置された場合，設置されていない場合のシミュレーションを行い，シミュレーション結果を比較していくことで，誘導灯の有効性を確認し，さらに参考のモデルをフロアフィールドモデルに変更する際の問題について確認した上で問題なしと判断し，フロアフィールドモデルの拡張によるシミュレータ高度化の可能性を示唆した．

目次

第１章　研究背景と目的

都市の過密化が進む現代においては，地下空間の有効利用が必要と考えられ，我が国でも地下街が少なからず存在し，建設が続けられている．この地下空間は，天候に左右されず自動車等と隔離された安全な空間である．一方，地下空間で一度火災等の災害が発生した場合，その閉鎖性から，地上空間にない特有の災害の拡がり方をし，人々に混乱によるパニック等が発生する．このため，地下空間独自の防災対策が必要となると言える．

上記の理由から地下空間の独自の防災対策の問題を解決していくためには，構造物の強度の見直しなどの，ハード面の対策だけでは不十分であり，避難計画や防災対策などのソフト面からの対策も必要となってくる．ソフト面の対策の一つとして，コンピューターを用いた群集の歩行シミュレーションというものがある．これは，現実では実測が困難な状態での歩行者の流れを視覚的に予測，分析できるなどの理由から有用な方法とされている．

歩行シミュレーションについてだが，いくつかの既存の技術が存在するので，以下に主なシミュレーションモデルについて示す．

① セルオートマトンモデル

対象とする空間を格子状の領域に分割し，複数の内部状態を持つ各セルの状態を時間とともに変化させることで，群集移動を表現する方法である．各セルがどのように時間発展していくかは，近傍のセルとの相互作用を規定するローカルルールに依存する．

②ネットワークモデル

空間をネットワークとして捉え，ネットワーク上を歩行者が移動するという大局的な立場からシミュレートするモデルである．ネットワークは線で表し，線上を移動する歩行者の交通量を調べることができる．

③流体モデル

群集流を連続した流体のように考え，連立方程式や運動方程式に基づいて解析しようとするモデルである．群集流をミクロの視点で見たときなど，歩行者の振る舞いを流体として考えた場合と一致するわけではないが，マクロな視点で見た場合などの大局的な把握を行うのに向いている．

④ポテンシャルモデル

歩行者と障害物に正の電荷，目的地に負の電荷を与え，荷電粒子間の相互作用を指すクーロンの法則という法則に基づき歩行者が目的地に向かって進むようにシミュレートするモデルである．歩行者と障害物に正の電荷，目的地に負の電荷を与えるため歩行者同士や歩行者と障害物などの組み合わせでは，両者の間に距離に依存する反発力が生まれ，歩行者と目的地の間には，引力が発生する．

⑤その他

その他，磁気モデルなどが存在する．

人間が避難する際の行動というものは，視界・性格・周囲の状況など様々な要因を考慮して行うもので，これらの要因に対応したモデルを作成していくには，上記した①～④のモデルの中では設定するローカルルールしだいでどんな要因も表すことができるセルオートマトン法が最適である．

セルオートマトン法は生物の自己複製機能を模擬するために，1940 年代末に数学者のウラムとフォン・ノイマンによって提案されたものである．その構造は単純であるが，様々な複雑現象を微分方程式等を用いずに簡易に表現可能であることから，自然現象等の離散的要素が並列的に作用する系のモデルとして幅広く適用されてきた．避難行動という群衆の流れにおいても微分方程式等では表現できない部分が生じ，また個々の人間が相互に影響を及ぼし合うので，局所的な相互作用に基づいた計算手法であるセルオートマトン法を適用することは有効と考えられる．

本研究ではセルオートマトンモデルをもとに作成されたモデルを参考にし，移動先の決定時の計算方法という点で別の手法を組み合わせるべく改変を行った．

2章では既存技術についての知識として，セルオートマトンの基礎である一次元セルオートマトンの説明と，二次元セルオートマトンの説明を行った．その後，本研究で取り上げた参考モデルについて述べ，作成したモデルについての知識としてフロアフィールドモデルについて述べた．

3章では提案手法について述べ，さらに2章で紹介した参考モデルについて述べた．その後フロアフィールドモデルである作成モデルについて述べ，シミュレーションを行い得られた特徴的な事柄を参考モデルと作成モデルとで比較しながら考察を行った．

第２章　既存の技術について

セルオートマトン（Cell Automaton）はごく単純な規則から予測もできない複雑なパターンを生成できる数理モデルであり，生物の組織形成モデルや複雑な自然現象のモデル化などに幅広く用いられている．以下にセルオートマトンの基本的な事項について記す．

2.1 一次元セルオートマトン

セルオートマトンの中でも最も単純な一次元セルオートマトンという例を用いて，その基本原理を示す．

ある帯状の図形を長さ方向でＮ個に分割し，分割された各々の図形をセルと呼ぶことにする．これらのセルについて，一方の端から1，２．．．Ｎというように番号をふっていき，それぞれのセルが0または1の状態をとるとする．

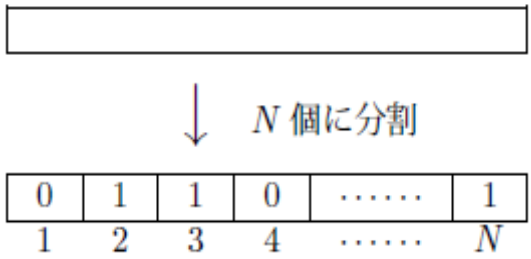

図1．一次元のセルの分割

各セルの状態は，離散時間単位毎に更新される．更新はセル空間全体で同時に行われる．更新後も更新前と変わらず，各セルは状態0か状態1のどちらかをとるが，どちらの状態をとるかはひとつ前の離散時間ステップにおける自分自身と周りのセルの状態など，自身を含む周囲のセルの状態によって定まる．このような局所的な相互作用に基づき，自然と組織が形成されることを局所近傍則という．すなわち，ｋ番目のセルが時刻ｔ＋１に0または1になるかは，時刻ｔでのｋ－１，ｋおよびｋ＋１番目のセルの状態によって決定されるということになる．

これを式で書くと，

$$a_k^{t+1} = f(a_{k-1}^{t}, a_k^{t}, a_{k+1}^{t})$$

と表現できる．ただし，a_k^{t}は時刻ｔにおけるｋ番目のセルの状態である．ｆはセルの時間発展のルールを表わし遷移則と呼ばれる関数であり，a_{k-1}^{t}，a_k^{t}，a_{k+1}^{t}の3つの変数がそれぞれ0または1の2つの状態を取り得ることから8種類の入力パターンに対して関数出力値，すなわち8つの写像を定義する必要がある．これら8つの遷移則をすべて一纏めにしたものをルールという．このルール自体は局所的なものだが，すべての空間においてこの近傍則は適応されることで次の時間ステップでの状態が一意に定まる．8 つの遷移則がそれぞれ 0 または1を出力するので，ルールの総数はは 28 = 256 通りある．遷移則の出力 8 つを並べて書き，2 進数として解釈すると 010110102 = 90 となる．

これより，この例は「ルール90」と呼ばれる．ここで問題になってくるのは両端のセルをどう処理するかということである．両端のセルはそれぞれ右端と左端で隣り合うセルが一つづつ欠けた状態となっており，ルールを適応することができないのである．

この問題の対策法がいくつか存在するので例を挙げる．

・両端のセルの外側に仮想的にセルを置く手法
・セル1とセルNとを輪のように両端で繋げて考える手法
・注目するセルの外のことは一切考えない手法
・両端の処理を確率に任せる手法
・etc

a_{k-1}^{t}	a_{k}^{t}	a_{k+1}^{t}	a_{k}^{t+1}
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

図2：ルール90一覧

両端のセルを繋げて考える「周期境界条件」と呼ばれる手法を用いて，ルール90を適用したものを時間発展させてみる．

まず初期状態として中央のセルの状態を1（黒），残りのセルの状態を0（白）とおく．そしてルール90に従い発展させていく．発展させていくと，シェルピンスキー・ギャスケットと呼ばれるフラタクル図形が描かれる．フラタクル図形とは，図形の一部を拡大すると同じパターンが再帰的に現れる性質（自己相似性）を持った図形のことである．図2がルール90を体起用したものの時間発展した図である．黒の部分が1で白の部分が0をあらわしている．初期状態は前述したとおり真ん中だけが1で後はすべて0である．できあがった複雑なフラタクル図形が生まれるというのは初期状態からは予想できないことである．

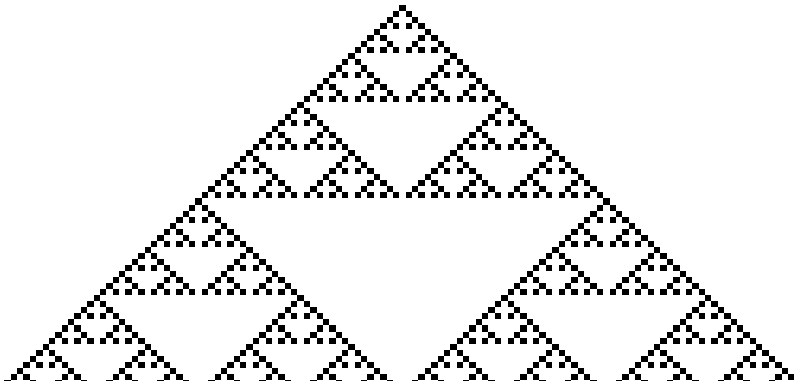

図3. ルール90の時間発展

図3の例のようにセルオートマトンはごく単純なルール設定でも時間経過によって複雑なパターンを生成できる. その時間発展の様相は4つのクラスに分類できるので, 以下に記す.

クラス1:すべてのセルが同じ状態で覆われ変化しなくなる

クラス2:周期的なパターンの繰り返しになる

クラス3:セル全体がランダムな時間発展をする

クラス4:長い過渡状態を持ち時間発展は複雑で局所化されている

クラス1とクラス2は秩序的に時間発展し, クラス3はカオスであり, 初期状態から先の状態を予測するのは困難である. カオスとは, 初期条件・境界条件を定めると以後の運動が決まるような簡単な系であっても、初期条件のわずかな差で大きく違った結果を生ずるような現象のことで, 気象現象・乱流や生態系の変動などに見られる自然界の法則性 を表す語である. クラス4は規則的な部分と不規則な部分が混在するパターンが展開される秩序とカオスの中間に位置する「カオスの縁」と呼ばれる発展をするクラスである. このカオスの縁こそが生命現象などの現実世界で見られる複雑な現象を引き起こす元だといわれている.

2.2.1 二次元セルオートマトン - ライフゲーム

前項ではセルオートマトンにおいて最も簡単な一次元セルオートマトンについて述べた. 本項では一次元から次元を一つ上げた二次元セルオートマトンを古くから有名なライフゲームという例を用いて紹介する.

ライフゲームは, 生命の繁殖を模したセルオートマトンモデルの一つである. セルは二次元平面で区切られ格子状に並んでいる状態で, セルの状態を0と1を用いてそれぞれ生と死を表すものとする. ここでは0を「死んでいる状態」, 1を「生きている状態」とする. セルの時間発展は, モデル内のとあるセルについて注目したとすると, 注目するセルの上下左右斜めに隣接する8つのセルの状態を確認しルールを適用を適用することで次の時間ステップの状態が定まる.

適用するルールを以下に記す.

- 現在生きているセルは, 隣接しているセルの中で生きているセルの個数が2個か3個であれば, 生存し続ける(維持)

- 現在死んでいるセルは, 隣接しているセルの中で生きているセルの個数が3個ならば, 生きた状態に変わる(誕生)
- それ以外の状態であれば, 死んだ状態になる(死亡)

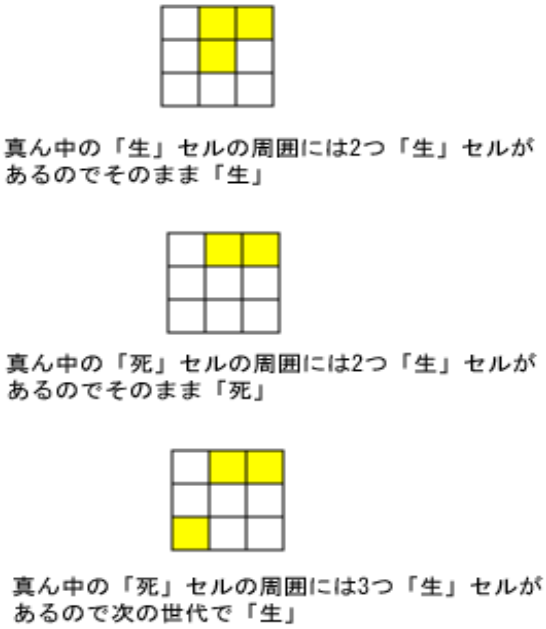

図4, ライフゲームのルール

自分の周りに生命が多くても少なくても死んでしまうというルールになっていることが分かる. ルール自体はとても単純なものであるが, 二次元のセル空間上において極めて複雑で面白い時間発展をすることが知られている.

ライフゲームはとても単純な規則の上に成り立ちながら, その単純な規則から様々な図形が生み出される. その非常に複雑な変化の振る舞いがまるで生命のように感じられる. このように単純な規則から複雑な振舞いが生じることは人工生命の分野でを創発と呼ばれる. ライフゲームは人工生命分野の古典的なプログラムといえるものであると言える.

2.2.2 二次元セルオートマトン - モデル化

ここまで二次元セルオートマトンをライフゲームという例を用いて説明してきたが，ここでははセルオートマトン法を用いたモデル化について説明する．

まず対象領域とする空間を一様に格子状に分割する．各格子は有限の状態を持つセルとする．あるセルの状態は自身を含む周囲8セル(ムーア近傍)の状態によって定まる．このように局所的な相互作用に基づいて，自然と組織が形成されることを局所近傍則といい，この局所近傍則に基づいた計算手法をセルオートマトン法という．セルオートマトン法による解説手順を図 1 に示す．

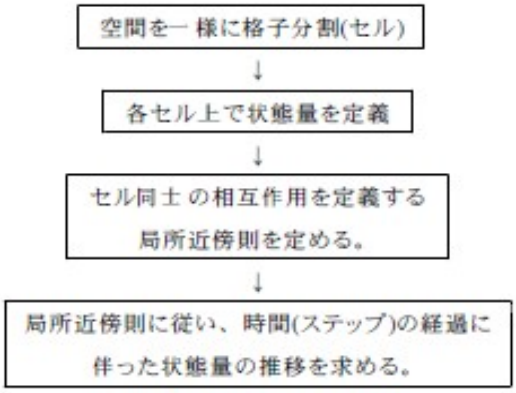

図5．モデル化のフロー

対象とするものをモデル化する際には図5のような手順でモデル化を行う．どんなに複雑なセルオートマトンでも基本的にはこのフローにしたがってモデル化される．ここで一番重要なのは局所近傍則である．この近傍則の定め方によって時間発展での変化の仕方が定められるからである．

2.3 人の流れシミュレーション

以下に二次元セルオートマトンの例としてセルオートマトンを用いたモデルについて述べる. ここで述べるモデルは本研究で参考にしたモデルであり, 実験結果や比較検証については後述するものとする.

2.3.1 セルオートマトンを用いたモデル

参考にした論文は, 地下空間での避難行動のシミュレーションを目的としたもので, 実在の地下街をモデル化し, シミュレーションを行ったものである. この論文では, 人間の行動決定の際に影響する要因である状態量と, それらの要因を組み合わせることでそのセルへの移動のしやすさを表した状態量の総和の計算によって移動を決定している. まず, 当該のセルに位置する人間が進行する際に支障がない最も進みやすく一番近い出口を判断する. 次にその出口を目標に状態量, 状態量の総和の計算を行い, その人間の周囲のセルで総和が最も小さいセルに移動を行う. そして, 1時間ステップの後, 再び一番進みやすい出口の判断と状態量の総和の計算を行い, 最も低い隣接セルに移動を行う. この行動を繰り返すことで, 人間の避難行動を再現している.

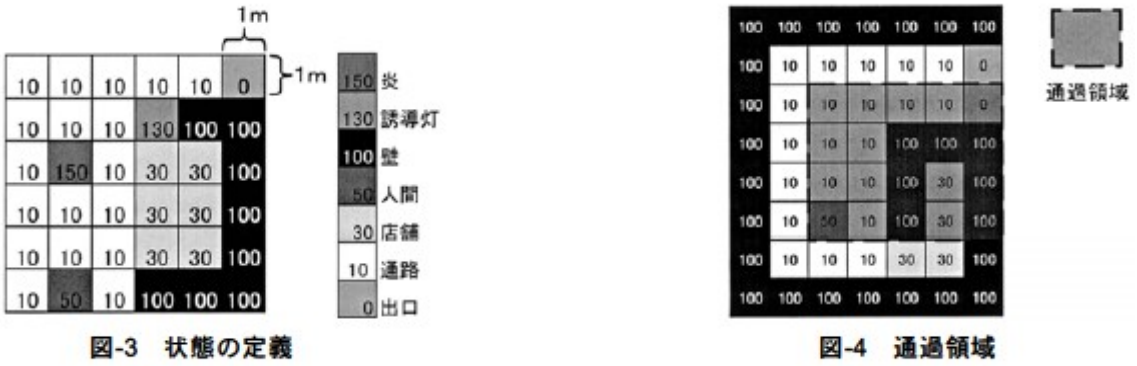

図-3 状態の定義　　図-4 通過領域

図6. 参考論文からの引用図

2.3.2 モデルの状態量

以下に参考にした論文で用いられた状態量について述べる.

- 出口までの距離の状態量
  最短の道筋で移動するという考えから, 出口までの直線距離を状態量として与える.

- 障害物周りの状態量
  一般的に, 避難の際には, 壁側などの障害物が多い空間よりも, 開けた空間に向かって移動すると考えられる. この表現のため, 状態量として障害物に隣接するセルと, 曲がり角の周囲のセルには＋の状態量を与えている. このモデルでは移動先のセル決定時に, 状態量の総和が低いセルを移動先として決定するため, ＋の状態量を与えるということは移動先として選ばれにくくするという意味がある.

- 堂々巡りを防ぐ状態量
  一般的に人間は一度通過した場所は記憶に残るため, 必要がなければ再度通過しないと考えられる. そのため, それぞれのセルは人が一度通過するたびに＋の状態量を与えられる.

- 環境を表す状態量
  周囲の状況を見て, 障害物が多く存在する方向には移動しないという考えから, 2ステップ先までの周囲8セルの状況の状況を調べ, それに応じた状態量を算出することで周囲の環境を評価する.

  以上の4つのセルが状態量として与えられている. それぞれの状態量の感度を決定する係数を乗じて加算したものが, 状態量の総和である.

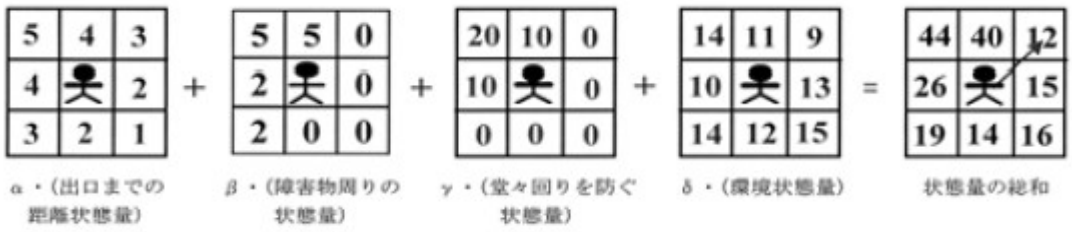

図7. 参考モデルの状態量計算の図

また，前記の4つの状態量以外にも，避難誘導灯にも状態量が与えられているのでそれについて記す．

一般的に災害時に人間は心理状態が不安定になり，通常時には容易であった袋小路からの脱出が困難となると考えられる．そのために，出口への道のりを示す役割と出口の場所を示す役割を持つ避難誘導灯が現実では多数設置されている．この避難誘導灯がもたらす避難行動への影響力を表現するために，参考論文では「避難誘導灯が示す順路」の状態量と，「避難誘導灯までの距離」の状態量の追加を行っている．まず，人間からもっとも近い誘導灯を選定し，その誘導灯が一定の距離以下である場合，（つまり離れすぎていて誘導灯が示す道筋が正しいと限らない場合があるのでこのような制限をつけている．）誘導灯が指し示す方向，例として誘導灯が左を向いていた場合，人間の周囲8セルの中で，左上，左，左下といったように左方向の要素を持つ方向に状態量を加える．状態量を加える際には状態量の値に負の係数を乗じてから行う．これにより誘導灯が示す方向のセルの状態量の総和が減少し，人間はその方向へ移動しやすくなる．

2.4 セルオートマトン・フロアフィールドモデル

セルオートマトン法を用いたフロアフィールドモデルについて特に人の流れについてを，例を用いて述べる． フロアフィールドモデル（Floor-Field Model）とは，床の上という場を人が動き回るような交通モデルのことを言う．

説明のための例として，図8のような二次元セルオートマトンを用いてモデル化した居室を考える．格子状に区切られた二次元空間を居室と考え，各セルに出口や障害物，人といったものを配置することで，現実世界の居室を模擬することが出来る．出口や障害物，人はセルに状態量として持たせることで表現する．図8では人が居るセルを1，出口を2，障害物を3と定めている．これである時間ステップの居室の状態を表すことが出来る，次の時間ステップに進むにあたり，人をどのように進めるかについてだが，人は移動を決定する際に，単純な二次元セルオートマトンモデルのように，近傍則に基づくルールのみによって動くわけではなく，他の要素によっても行動を左右されるものである．例えば火事の際に炎から避けようとするように動こうとするものであったり，人の心理要因に起因するものであったり，誘導効果を持つものの影響であったり，人は人以外のものからの影響を受け行動を決定するものである．これらの人の性質であったり，外部要因であったりなどの性質をモデル化する手法として，フロアフィールドモデルが知られている．

フロアフィールドモデルは，まず各セルに例として挙げた図8の居室のように，人や障害物，出口といった状態量とは別に，フロアフィールド（以下 FF と記述）という状態量を持たせる．そして人は，自分の居るセルの周囲のセルの FF を比較して移動する方向を決定する．この移動先の決定の際には数式等や確率を用いて移動先の決定を行う．

FF は静的 FF と動的 FF の二つの FF からなる．例としてあげた居室であるが，本研究は防災を目的としたものであるので，例としてあげているものも，特に断りを入れない場合は人は避難を行うために，脱出を目指して初期状態の位置から出口に向かって動くものとする．

0	0	出口(3)	0	0
人(1)	0	0	0	0
0	障害物(2)	障害物(2)	障害物(2)	0
0	0	0	0	人(1)
0	人(1)	0	0	0

図8. CAによるモデル化の例

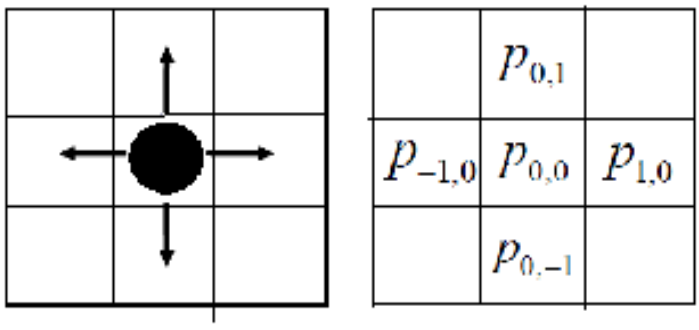

図9．移動可能なセルと移動確率

2.4.1 静的 FF

人が自分で設定した目的地に向かって移動する場合，特別な事情がなければ最短距離を通ると考えられる．これを反映したルールが静的 FF である．具体的には各セルに目的地までの距離を持たせ，人がセル間を移動するときは各セルを比較し，静的 FF が減少するセルに移動しやすくする．静的 FF は，目的地までの最短距離というものが，時間的に不変であることから静的 FF といわれている．図11のようにセルに名前をつける．セル O からセル X(X = U; D; R; L) に人が静的 FF によって動く確率を，

$$ps(O \rightarrow X) = \exp\{ks(S\,O - S\,X)\}$$

とおく．ただし，

- $ks \in (0,1)$:静的 FF の調節パラメータ(この値が大きい程静的 FF の影響が大きい)
- $S\,z(Z = O,U; D,R; L) \in ([0,1)$:セル Z の静的 FF(出口からの距離)

とする．

2	1	出口	1	22
√5	√2	1	√2	√5
2√2	√5	2	√5	2√2
1+2√2	壁	壁	壁	1+2√2
2+2√2	3+2√2	4+2√2	3+2√2	2+2√2

図10. 出口からの静的FF量

0	U	0
L	O	R
0	D	0

図11. 上下左右の対応

2.4.2 動的 FF

二つ目の FF は動的 FF といい，これは人の足跡を表す．蟻がフェロモンを介して相互作用するように，人は足跡を介して相互作用をするとする．蟻が自分のいた場所にフェロモンを残すように，人は自分の1 ステップ前にいたセルに足跡を残すとする．そして蟻がフェロモンの多い場所に行くように，人はより足跡の多い場所へ動くようにする．この足跡が動的FF である．動的FF による移動確率を，

$$pd1(O \rightarrow X) = \exp(kd1D1,X)$$

とする．ただし，

- $kd1 \in (0;1)$:動的 FF の調節パラメータ(この値が大きい程動的 FF の影響が大きい)
- $D1,X(X = U, D, R, L) \in (0,1)$:セル$Z$ の動的FF. 足跡の数を表す.

とする．

また，人が残した足跡は時間とともに拡散し，減衰もする．拡散率を α，減衰率を δ と定めると，拡散による動的 FF の変化は，

$$D_{ij}^{t+1} = (1-\alpha)D_{ij}^{t} + \frac{\alpha}{4}(D_{i+1,j}^{t} + D_{i-1,j}^{t} + D_{i,j+1}^{t} + D_{i,j-1}^{t})$$

となり，また減衰による動的 FF の変化は，

$$D_{ij}^{t+1} = (1-\delta)D_{ij}^{t}$$

となる．これらをあわせると，

$$D_{ij}^{t+1} = (1-\alpha)(1-\delta)D_{ij}^{t} + \frac{\alpha(1-\delta)}{4}(D_{i+1,j}^{t} + D_{i-1,j}^{t} + D_{i,j+1}^{t} + D_{i,j-1}^{t})$$

という式が導かれる．これは拡散を先に計算しても，減衰を先に計算しても，結果が変わらないことを示している．

第３章　提案手法

この章では本研究で行った手法に関することを記す．本研究では２章で記した二次元セルオートマトンモデルを元に，局所的な避難誘導灯に注目したモデルにフロアフィールドモデルの確率計算法を組み合わせたものを用い改変を行った．以下にそれについて記す．

3.1 参考モデルの問題点

参考とした先行研究では図12に示すような2つのケースに分けた実験を行っている．

一つ目のケースは，避難誘導灯を人間に認識させた場合で，二つ目のケースは，誘導灯を認識させない場合である．ここでは特別な災害等は考慮されておらずただ脱出のみを目的としている．

引用した図を見ると，誘導灯を設置したものはそれに従い迅速な避難が行われているが，誘導灯を設置していないものは，出口に対して進行する傾向が強いあまりに，脱出できずにいることが確認できる．

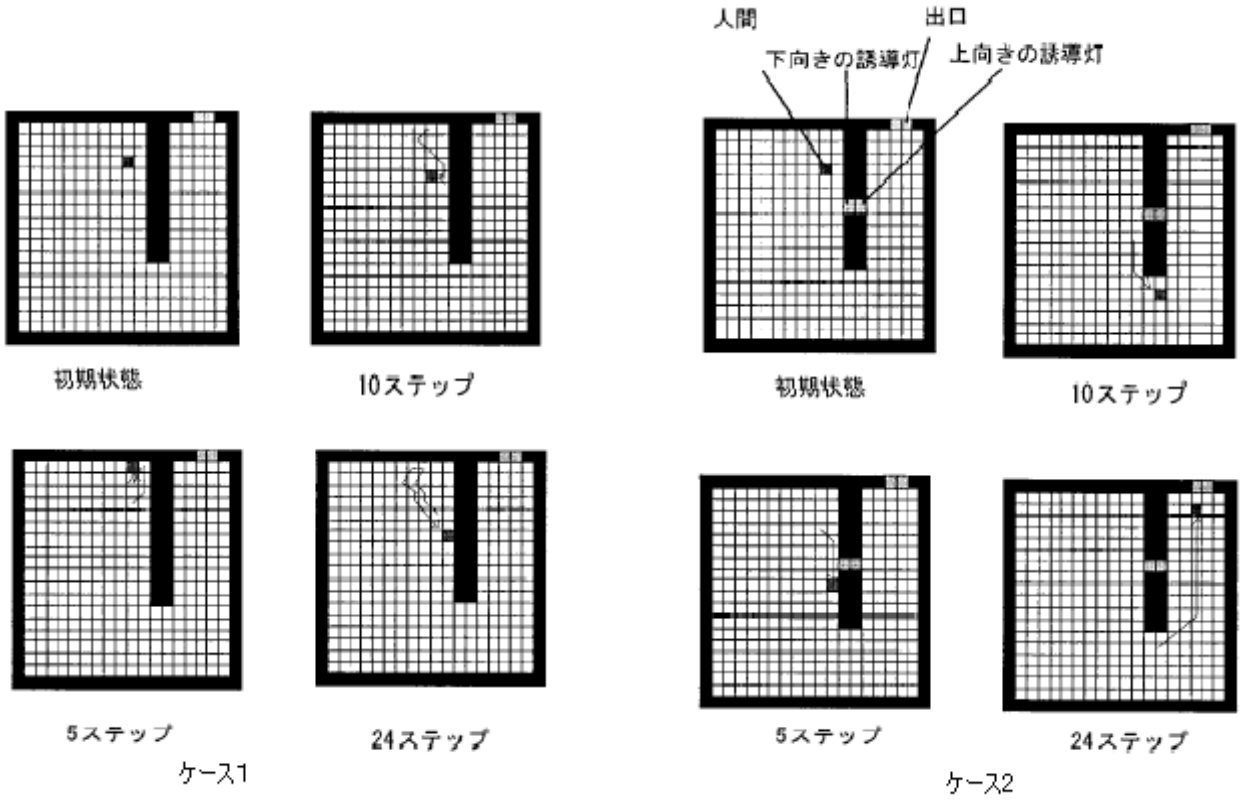

図12.時間ステップ経過図（ケース1：誘導灯あり，ケース2：誘導灯なし）

参考モデルは状態量の総和のみで行動決定を行うモデルであり，このモデルを用いたシミュレーションの結果は図 12 に示した．結果を見るに，誘導灯の有無が脱出の成否に関わることが分かる．そのことから脱出時における誘導灯の有効性というものが確認できる．他に特徴的な要素として，参考モデルで誘導灯を用いた脱出を行う際，常に最短のルートを通り，同一の脱出場所からシミュレーションを何度行っても同じルートを通るというのが確認できた．

参考したモデルのように，フィールドの中にいる人の数が一人で，移動先決定のための状態量が少ない場合では気にすべき事柄も少なく，問題も起こりにくい．しかし，モデルを拡張し，高度化しようとするとなると問題も多い．例えば単純に避難する人の数が増えたとすると，一人のときでは観測できなかった人の流れが観測できるようになる．セルオートマトン法の性質上，1時間ステップで複数の人をパラレルアップデートで同時に移動させようとすると，同じセルに二人以上の人が入ろうとし衝突が起こることがある．この問題は，確率を用いたパラメーターを導入してやり，一方が進むか待つか両方動かないか，などといったようにしてやることで，行動を表すことができる．確率を用いてやらずに，衝突状態の解決を常に同じ解決方法にしているモデルは，人の動きを扱うシミュレータとして重大な問題を抱えていると言える．確率を用いてやることで行動決定に振れ幅ができ，毎回違った行動を行い，その結果が全体に影響を及ぼしていくことが可能になる．他にも確率を用いてやると，前記した同じセルへの移動や，出口付近の渋滞状態の解決など，いわゆる競争・協力状態を表すことができたり，移動可能なセルが周囲 8 セルになったときや，急いでいる状態など，人間の心理状態と移動速度の要因を考えるときにどれだけのセルを移動するか，などといった状態も表すことができるようになる．確率を用いない状態量の総和の計算のみで行動決定を行うモデルでは前記した問題を解決していくのは難しい．

次に，参考モデルを改変した提案モデルについて考えてみる．最も大きな違いは，移動先の決定のために行われる計算方法と，最終的な移動先の決定方法である．参考モデルは，状態量を計算して求められた状態量の総和が絶対で，それを覆して移動を行うことはない．一方提案モデルでは，状態量の総和を求める点までは同一だが，最終的な移動先の決定は確率を用いて行われる．つまり，移動しやすいという指針はわかっても移動先に幅が出来るということである．
前述したように，状態量の総和のみの計算方法では，シミュレーションを行う際に特定の場面で同一の行動を取ったり，常に最短のルートを選択したり，など正しすぎるという問題が出てくる．そこで，シミュレーションの高度化のために参考モデルをフロアフィールドモデルを用いた提案モデルに置き換え，より人間らしく，高度にしていくのが本研究の目的である．

以上を踏まえて前述したこの参考モデルの状態量を見直してみるが，出口までの距離の状態量というのはそのまま静的 FF に置き換えることが出来る．障害物周りの状態量というのは変更は必要ないと思われるので，用いる値のみを変更してそのまま用いればいいと考えられる．次に堂々巡りを防ぐ状態量であるが，これは必要ないと考える．前述したとおり，フロアフィールドモデルでは，移動を行う際に移動先がぶれる場合がある．同じ場所を回り続けるという問題は新たな状態量を定義してやらなくても，確率によって発生する移動先のぶれで解決されるのではないかと考えるため，これは改良時には廃止するものとする．最後に，環境を表す状態量であるが，これはつまり動的 FF の拡張にあたるものであると考える．周囲8セルを確認するだけではなく，人間の視野を表すために参照する周囲のセルを増やしている．これもフロアフィールドで置き換える際には必要だと考える．ただし，今回は参照するセルが周囲 8 セルではなく，簡易の周囲 4 セルであるため，この状態量は削るものとする．実際に改良モデルを作成し，参考実験と同じく二つのケースで実験を行い，それぞれ比較してやることで，二つの手法を組み合わせることによって起こる変化を確認してやることが必要である．

3.2 作成するモデルについて

実験のために作成するモデルについて記す.

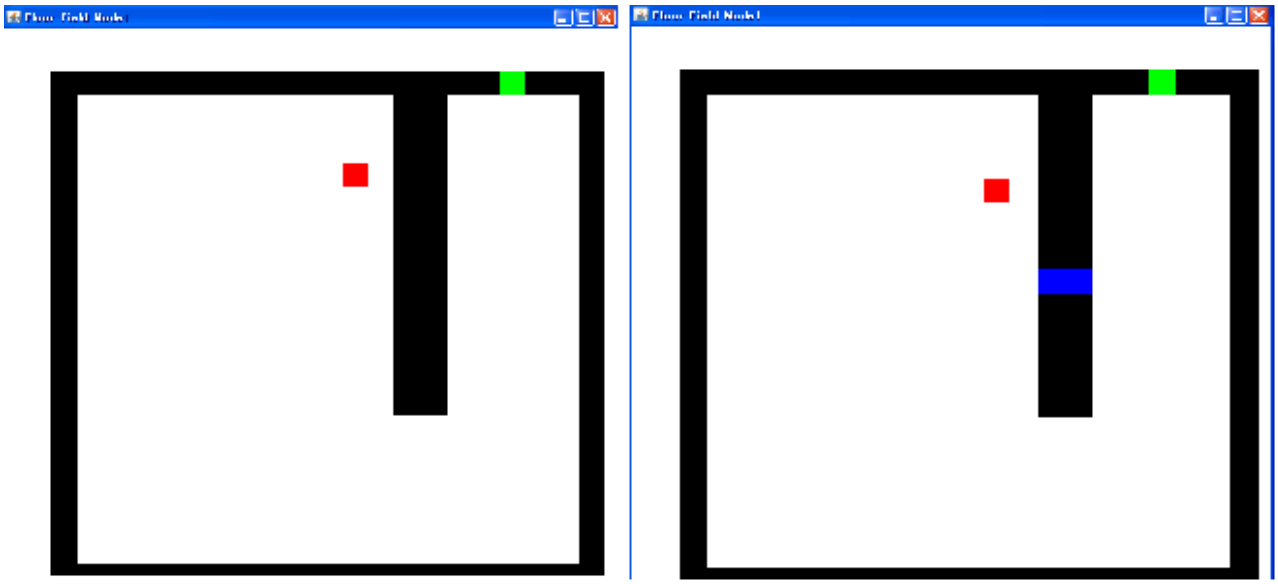

図13. 作成モデル図(左:誘導灯なし, 右:誘導灯あり)

図13のようなフィールド上に人を配置しシミュレーションを行う. 左が誘導灯を設置していない図で, 右が誘導灯を設置した図である. それぞれ横 20 セル×高さ 20 セルの広さを持つフィールドである. 初期状態の人の位置は, ランダムではなく固定した. また, 誘導灯の効果がわかりやすいように, 初期位置は, 袋小路の角の部分に近い, 誘導灯の効果範囲の中に設定した.

人は移動する際に自分の周囲 4 セルを参照し, いずれかに移動する. 人自身に視界を持たせていないので, 誘導灯の周囲何セルかに誘導効果を持たせることで表現した. 効果範囲は誘導灯の周囲 6 セルとした. 袋小路側の誘導灯は下方向への誘導効果を持ち, 出口側の誘導灯は上方向への誘導効果を持つ. 図14に誘導灯の効果範囲について視覚的に示す.

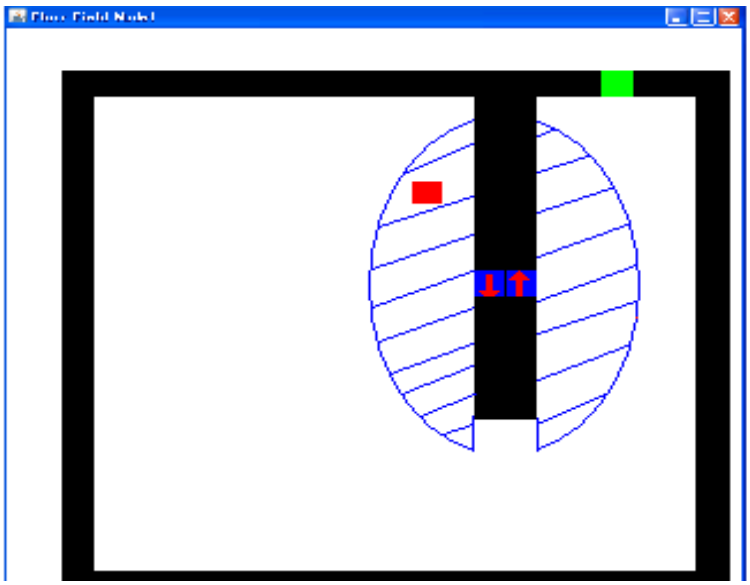

図14. 誘導灯の効果範囲

3.3 シミュレーション

シミュレーションを行い、その経過と結果を観察し，得られた特徴的な事柄を参考モデルと作成モデルとを比較しながら記す．

3.3.1 シミュレーション結果

シミュレーションを行った際について記す．図15は作成モデルの誘導灯を設置していない場合のシミュレーション結果の一例を視覚的に表した図であり，図16は作成したモデルの誘導灯を設置した場合のシミュレーション結果の一例を視覚的に表した図である．図15と図16はそれぞれ図12のケース1とケース2に対応している，また，図15と図 16 で例に取ったシミュレーション結果は，複数回行ったシミュレーションの中から提案手法であるシミュレーターの特徴をよく表していると思われる結果を例に取ったものである．

誘導灯の効果範囲は誘導灯の周囲 6 セルに限定している．その範囲の外では静的FF以外に移動に効果を及ぼすものはない．時間ステップ毎の移動の様子を表すのは難しいので，視覚的に移動の様子を矢印で表す．人の初期配置位置は固定されていて，常に同じ位置から開始される．

まず誘導灯がないモデルについてだが，図12のケース2と図15を見て分かるように，これらのモデルはシミュレーションが開始されると出口まで最短で進もうとする性質から壁の存在を考えず袋小路の角の部分に進んでいった．その後，袋小路の角付近を行き来するのみで，自力での袋小路脱出はついに果たせなかった．図12ケース2のモデルは移動が周囲 8 セル可能であるので視覚的に直線で表した場合移動量が大きく見える．対して図15の作成したモデルの方は移動可能なセルが周囲 4 セルなことから袋小路の角を含む隣接セルにいる時間が多くなっている．細かな差異はあれど，両者ともに袋小路からの脱出はできていない．このことから参考にしたモデルの誘導灯を設置しなかった場合と作成しシミュレーションを行ったモデルは同じ結果に終わっているということが分かる．

次に図12ケース1と図16のような誘導灯を設置した場合について記す．両モデルともにシミュレーションが開始された後の傾向として，まず誘導灯の誘導方向に従って進み，そのフィールドをさえぎる壁を越えた後に若干のランダムさを持ちながら移動し次の誘導灯効果範囲内のセルまでたどり着き，出口まで無事にたどり着く，という傾向が確認できた．また，細かな挙動として，図16で表される作成モデルは移動先の選択にランダムさが付加されるその特性から，誘導灯効果範囲において，出口方向以外への移動を行うことで，ケース1の参考モデルの例のように無駄なく避難行動を行わないということがあった．しかしその挙動による脱出が大幅に遅延されることはなく，最終的には脱出をが成功していることがほとんどであった．このことから図12のケース1，参考モデルの誘導灯を設置した場合と図16の誘導灯を設置した作成モデルは，同じ結果に終わっているということが分かる．

以上，作成モデルの誘導灯を設置していないモデルは，移動先がランダムさを持っているといっても移動先は出口方向がある上と右に移動しやすいというのは変わっておらず誘導灯のないモデルは自力で袋小路を脱出するには至らなかった．　一方，誘導灯のあるモデルの方は，おおむね狙い通りの動きをした．ただ，誘導灯の効果範囲セル外では，移動にランダム性が付加されるため，シミュレーションを行う毎に脱出までの経過時間ステップ数に幅ができることが観測できた．

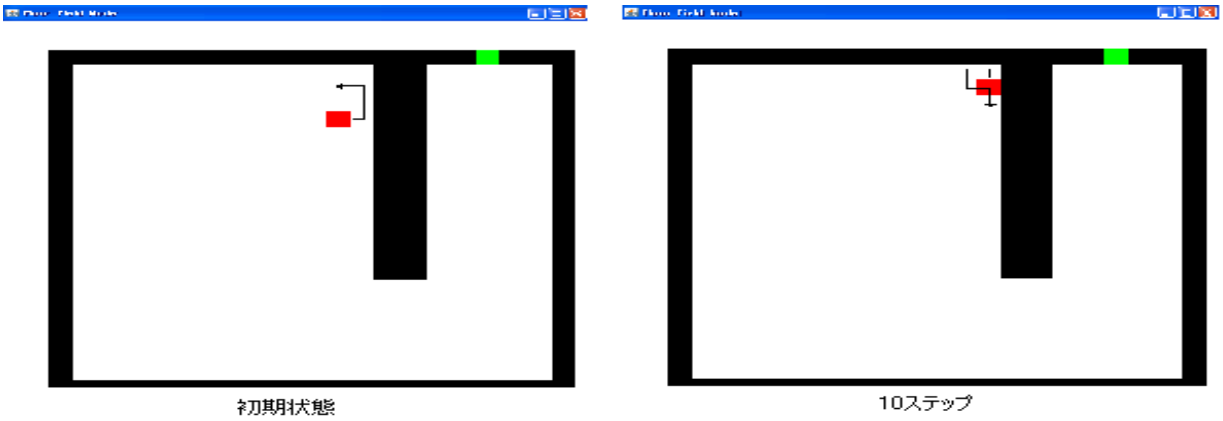

図15.ケース1誘導灯非設置のシミュレーション結果

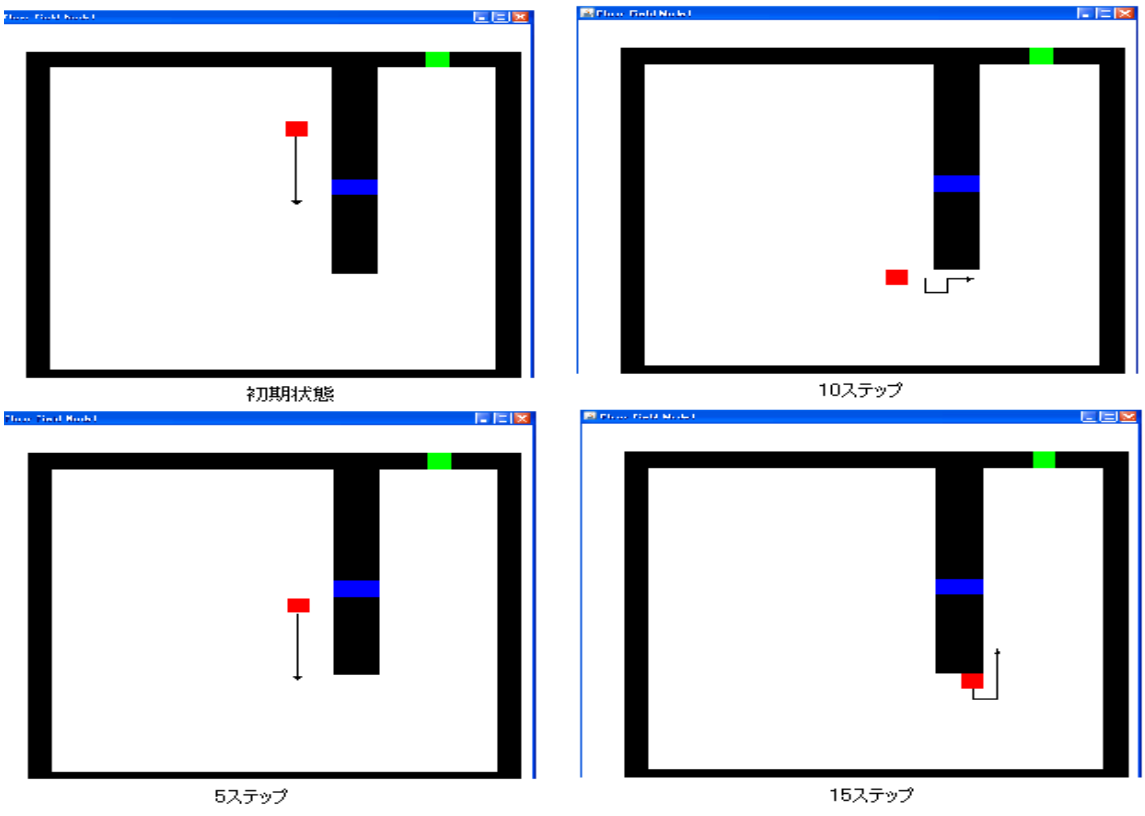

図16.ケース２誘導灯設置時シミュレーション結果

3.3.2 考察

シミュレーションから得られた結果について考察と分析を行う. 参考モデルに比べ作成モデルはシンプルに作られているため細かなところで差異が発生することがある.

作成したモデルで行ったシミュレーションについてだが, 避難誘導灯が設置されたモデルと設置されていないモデルでは, やはり誘導灯が設置されたモデルの方は参考モデルと同じく, 袋小路を脱出でき, 避難を行っていることが分かる. 設置されていないモデルは, 参考にしたモデルと同じく, 袋小路の角の部分をうろうろするばかりで脱出には至らなかった. シミュレーションを何度も行って観察してみたが, 角の部分から5セルも離れることはなかった. 参考モデルの誘導灯を設置していないモデルも同じく袋小路内で手間取り脱出できていない.

誘導灯を設置した場合の作成モデルだが, このモデルでは, 人が移動可能なセルが周囲 4 セルである. 図 12 ケース1のように, 周囲 8 セルに移動が可能だったならば誘導効果は下の要素を含む下 3 方向に誘導効果を持ったものになり, よりスマートな避難が行われただろう. その移動可能セルの問題を差し引いても, 全体的に見て参考にしたモデルよりも時間がかかるケースが多かったように見受けられる. その原因は取り入れたランダム移動の部分だと思われる. 図16の一例の 10 ステップ経過の部分のように誘導灯の効果範囲セルを脱した後に僅かであるが, ランダム移動で出口以外の方向に向かってしまうというケースが起こり, その結果わずかなステップ数の差ではあるが, 脱出まで時間ステップ数に差ができてしまっている. まとめると, 参考モデルはどんなときも直線的に進むのに対して, 作成モデルはシミュレーションによってはやや蛇行して進んでいく場合もある. この蛇行が原因で避難完了までの時間ステップに幅ができてしまうのだが, これは問題がないと考える. なぜなら, 避難完了までの時間ステップの少なさそのものがシミュレーションの評価につながるわけではないからである. ここで重要なのは参考モデルで可能だったことが作成モデルでも可能かどうか, という点である. 今回のような人間が一人, という状況では分からないが, 多くの人が一度に避難を行うような大規模なシミュレーションなどでは今回のやや蛇行気味の挙動のような脱出を目的とした無駄のない最短移動以外の行動というものが必ず必要となってくる. それは衝突の際の譲り合いであったり, 人間の心理に起因した行動決定であったり, 出口が分からない状態での分かれ道であったり, そういう状況が表れるような人の行動を表した高度なシミュレーションモデルを作成していくには, 確率移動を行うフロアフィールドモデルが有用であると分かるだろう.

以上より, シミュレーションの結果を観察し, 参考モデルと作成したモデルとでは, 誘導灯の有無によるそれぞれ対応したシミュレーションの結果に違いは起こっていないということが分かった. よって, セルオートマトンモデルをフロアフィールドモデルに変更を行っても問題はないものだと考えられる.

第 4 章　まとめ

本稿では，参考とする論文を元に避難行動シミュレーションを構築し，改変した上でシミュレーションを行い結果の観察を行った．

誘導灯については，設置した誘導灯はうまく効果を及ぼした．今回は移動先が周囲 4 セルのみなので誘導灯は単に下方向にしか誘導効果を持っていない．そのため，シミュレーション開始後，初期状態から単に下に向かうだけとなっている．

次に確率移動，つまりフロアフィールドモデルに変更したことによる移動先の決定方法の変更について述べる．今回作成したモデルは，参考にしたモデルと比べると挙動の特徴という点で異なる部分が観測された．参考モデルの直線的な最短移動的な挙動に対し，作成したモデルはやや蛇行的な挙動が特徴として観測された．ただしこの特徴というのは，設置された避難誘導灯の範囲外でのみ観測される．この特徴により，脱出完了までに要した時間ステップが全体的に多くなってしまった．しかし，この最短の脱出のための移動以外の行動こそフロアフィールドモデルで見られる挙動の特徴であり，作成モデルにおいてその特徴的挙動を確認することができた．

以上，本研究では，避難誘導灯が設置されている場合と，そうでない場合における群集の避難行動の表現のためにシミュレータを構築し，シミュレーションの結果と全体の挙動に注目することでフロアフィールドモデルへ変更する際の問題と変更することによる有用性を確認した．避難行動シミュレーションの高度化を図っていくならば，人間自体に視野を持たせるためのFFや，煙などの視野を狭めるものの存在を表すFFなどのFFの拡張を行っていく必要がある．拡張されたFFを用い高度化されたシステムを用いれば都市化の進む現代において重要視されている地下空間の防災対策の高度化を図ることが期待できるだろうと考えられる．

謝辞

本論文を作成するにあたり、指導教官の三好教授から、親身なご指導を賜りました．教授のご指導なくしては論文の完成には至りませんでしたここに感謝の意を表します．

また，意見などの協力を頂いた同じ三好研究室の皆様に感謝の意を表します．

参考文献

森下信著作(2003年出版)
セルオートマトン　複雑系の具象化
養賢堂発行

柳沢大地, 西成活裕
回り込みと壁・障害物の効果による人の移動流量の変化
http://www.riam.kyushu-u.ac.jp/fluid/meeting/18ME-S5/papers/Article_No_14.pdf

柳沢大地, 西成活裕
群集の集団運動と拡張フロアフィールドモデル
http://www.riam.kyushu-u.ac.jp/fluid/meeting/17ME-S2/papers/Article_No_28.pdf

松田泰治, , 塚久哲, 樗木武, 大野勝, 磯部淳志
火災および避難誘導灯を考慮した地下街における群集の避難行動シミュレーションに関する研究
http://www.fri.go.jp/ronbun/R2002/H14_S02.pdf

西成　活裕
セルオートマトンによる複雑現象のモデル化
http://park.itc.u-tokyo.ac.jp/tknishi/mypapers/rikouJ.pdf

付録　ソースコード

```
import java.awt.*;
import java.lang.*;
import java.util.Random;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;

public class FloorField01 extends JApplet
   implements ActionListener{
   // 描画のためのパラメータ
   static double frameWidth  = 680.0;
   static double frameHeight  = 680.0;
   final static double topMargin   =  50.0;
   final static double bottomMargin =  50.0;
   final static double sideMargin   =  50.0;

   final static Color bgcolor = Color.white;
   final static Color fgcolor = new Color(100,0,200);
   final static BasicStroke normalStroke    = new
BasicStroke(1.0f);
   final static BasicStroke thinStroke    = new
BasicStroke(0.1f);
   static double height,width;

   // エージェントの情報
   static int nAgent = 1;
   static int [] agentX = new int [nAgent];
   static int [] agentY = new int [nAgent];

   // ２次元セル
   final static int nCell    =  20;
   static int [][] cell = new int [nCell][nCell];
   static double dx,dy;

   // 出口の設定
   static int exitX = 17;
   static int exitY = 0;
   //
   static Random rand = new Random();

   String param; // パラメータ入力のためのバッファ
   static double ks = 1.0; // 近づく傾向の大きさ
   // コマ送りの間隔
   static int delay = 100;
   boolean frozen = false;
   Timer timer;

   AnimationPane animationPane;

   FloorField01() {
      for(int j = 0; j < nCell; ++j) {
         for(int i = 0; i < nCell; ++i) {
            cell[j][i] = 0;
         }
      }
   }

   // セルに nAgent 人を配置する
   void initCells(){
      double r;
      for(int i=0;i<nAgent;i++){
      //ランダム配置する場合は以下を用いる
      //      int x = rand.nextInt(nCell);
         //   int y = rand.nextInt(nCell);
            int x = 11;
            int y = 4;
            if(cell[y][x] == 2) {
               --i;
               continue;
            }
            agentX[i] = x;
            agentY[i] = y;
            cell[y][x] = 1;
      }
   }

   // 次のステップの計算
   void goNextStep(Graphics2D g2){
            int i,j,x,y,prevX;
            double qq,qr,ql,qu,qd,
               pr,pl,pu,pd,psum,r,
               px,py;
            Boolean moved = false;
            //System.out.println("goNextStep");
            for(i=0;i<nAgent;i++){
               x = agentX[i];
               y = agentY[i];
               qq = Math.sqrt((x-exitX)*(x-exitX) +
(y-exitY)*(y-exitY));

               qr = Math.sqrt((x-exitX+1)*(x-
exitX+1) + (y-exitY  )*(y-exitY  ));
               qd = Math.sqrt((x-exitX  )*(x-exitX  )
+ (y-exitY+1)*(y-exitY+1));
               ql = Math.sqrt((x-exitX-1)*(x-exitX-
1) + (y-exitY  )*(y-exitY  ));
               qu = Math.sqrt((x-exitX  )*(x-exitX  )
+ (y-exitY-1)*(y-exitY-1));

               pr = Math.exp(ks*(qq-qr));
               pd = Math.exp(ks*(qq-qd));
               pl = Math.exp(ks*(qq-ql));
               pu = Math.exp(ks*(qq-qu));

               psum = pr + pl + pu + pd;
               pr /= psum;
               pl /= psum;
               pu /= psum;
               pd /= psum;

               for(;;) {
                  r = rand.nextDouble();
                  if (r < pr){
                     if(x<nCell-1){
                        agentX[i] +
```

```
+;
if(cell[agentY[i]][agentX[i]] == 2) {
                                        --
agentX[i];
        continue;
                                    }
                                    moved =
true;
                                }
                        }
                        else if (r < pr + pd){
                                if(y<nCell-1){
                                    agentY[i] +
+;
if(cell[agentY[i]][agentX[i]] == 2) {
                                        --
agentY[i];
        continue;
                                    }
                                    moved =
true;
                                }
                        }
                        else if (r < pr + pd + pl){
                                if(x > 0){
                                    agentX[i] --;
if(cell[agentY[i]][agentX[i]] == 2) {
                                        +
+agentX[i];
        continue;
                                    }
                                    moved =
true;
                                }
                        }
                        else{
                                if(y > 0){
                                    agentY[i] --;
if(cell[agentY[i]][agentX[i]] == 2) {
                                        +
+agentY[i];
        continue;
                                    }
                                    moved =
true;
                                }
                        }
                        break;
                }
                if(moved){
        g2.setPaint(Color.white);
                                px = sideMargin + x *
dx;
                                py = topMargin  + y *
dy;
                                g2.fill(new
Rectangle2D.Double(px,py,dy,dx));
                                g2.setPaint(Color.red);
                                px = sideMargin +
agentX[i] * dx;
                                py = topMargin  +
agentY[i] * dy;
                                g2.fill(new
Rectangle2D.Double(px,py,dy,dx));
                            }
                        }
    }

    // 動かないパラメータの設定はここで行う
    void buildUI(Container container){
                        timer = new Timer(delay, this);
                        timer.setInitialDelay(0);
                        timer.setCoalesce(true);
                        animationPane = new
AnimationPane();
                        container.add(animationPane,
BorderLayout.CENTER);
                        Dimension d = getSize();
                        frameWidth  = d.width;
                        frameHeight = d.height;
                        width =  frameWidth - 2 * sideMargin;
                        height = frameHeight - (topMargin +
bottomMargin);
                        dx = width / nCell;
                        dy = height / nCell;
    }
    //Invoked by a browser only.
    public void init(){
                        param = getParameter("ks");
                        ks = Double.parseDouble(param);
                        setBackground(bgcolor);
                        setForeground(fgcolor);
                        buildUI(getContentPane());
    }
    //Invoked by a browser only.
    public void start() {
            //startAnimation();
    }
    //Invoked by a browser only.
    public void stop() {
            //stopAnimation();
    }
    public synchronized void startAnimation(){
                        //animationFlag = 0;
                        System.out.println("startAnimation");
                        initCells();

                        if(frozen){
                        }else{
                            if(!timer.isRunning()){
                                timer.start();
                            }
                        }
    }
    public synchronized void stopAnimation(){
                        //Stop the animating thread.
```

```
                System.out.println("stopAnimation");
                if (timer.isRunning()) {
                    timer.stop();
                }
    }
    // 以下がアニメーションの１コマに相当：ここで描画処理など。
    public void actionPerformed(ActionEvent e){
            animationPane.repaint();
    }
    class AnimationPane extends JPanel{
                // Draw the current frame of animation.
                public void paintComponent(Graphics g){
                    Graphics2D g2 = (Graphics2D) g;
                    int x,i,j;
                    // クリア
                    g2.setPaint(bgcolor);
                    g2.fill(new Rectangle2D.Double(0, 0, frameWidth, frameHeight));
                    // エージェントの描画
                    if(timer.isRunning()) {
                        goNextStep(g2);
                    }
                    // セルの描画
                    for(j = 0; j < nCell; ++j) {
                        for(i = 0; i < nCell; ++i) {
                            switch(cell[j][i]) {
                            case 2:
                                g2.setPaint(Color.black); // 壁
                                break;
                            case 3:
                                g2.setPaint(Color.blue); // 上
                                break;
                            case 4:
                                g2.setPaint(Color.cyan); // 左
                                break;
                            case 5:
                                g2.setPaint(Color.blue); // 下
                                break;
                            case 6:
                                g2.setPaint(Color.orange); // 右
                                break;
                            }
                            if(cell[j][i] >= 2 && cell[j][i] <= 6) {
                                g2.fill(new Rectangle2D.Double(sideMargin + i * dx, topMargin + j * dy, dx, dy));
                            }
                        }
                    }
                    // 枠の矩形を描画する
                    g2.setStroke(thinStroke);
                    g2.setPaint(fgcolor);
                    g2.draw(new Rectangle2D.Double(sideMargin,topMargin,width,height));
                    // 出口の描画
                    g2.setPaint(Color.green);
                    g2.fill(new Rectangle2D.Double(sideMargin + exitX * dx,
                                topMargin  + exitY * dy,
                                dx, dy));
                }
    }
    public static void main(String argv[]) {
                JFrame f = new JFrame("Floor Field Model");
                final FloorField01 controller = new FloorField01();

                // サイズを buildUI の中の getSize()に渡すため。
                controller.setSize(new Dimension((int)frameWidth,
                (int)frameHeight));
                f.setBackground(bgcolor);
                f.setForeground(fgcolor);

            controller.buildUI(f.getContentPane());
                // ウィンドウのボタンを押した時の動作
                f.addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent e) {
                                System.exit(0);
                            }
                    });
                // キーを押したときの動作
                f.addKeyListener(new KeyAdapter() {
                        public void keyPressed(KeyEvent e) {
                                if(e.getKeyCode() == KeyEvent.VK_ENTER) {
                                        if(!controller.timer.isRunning()) {
            controller.startAnimation();
                                        }
                                        else {
            controller.stopAnimation();
                                        }
                                }
                        }
                });
                f.addMouseListener(new MouseAdapter() {
                        public void mousePressed(MouseEvent e) {
```

```
                if(controller.timer.isRunning()) {
                    return;
                }

                int x = e.getX();
                int y = e.getY();

                if(e.getButton() ==
MouseEvent.BUTTON1) {
                    for(int j = 0; j
< nCell; ++j) {
        double h = topMargin + j * dy + 30;
        for(int i = 0; i < nCell; ++i) {
        double w = sideMargin + i * dx + 8;
        if(y >= h && y < h + dy) {
            if( x >= w && x < w + dx) {
                if(cell[j][i] == 2) {
                    cell[j][i] = 3; // 上
                }
                else if(cell[j][i] == 3) {
                    cell[j][i] = 4; // 左
                }
                else if(cell[j][i] == 4) {
                    cell[j][i] = 5; // 下
                }
                else if(cell[j][i] == 5) {
                    cell[j][i] = 6; // 右
                }
                else {
                    cell[j][i] = 2; // 壁
                }
            }
        }
                    }
                }
                else if(e.getButton()
== MouseEvent.BUTTON3) {
                    for(int j = 0; j
< nCell; ++j) {
        double h = topMargin + j * dy + 30;
        for(int i = 0; i < nCell; ++i) {
        double w = sideMargin + i * dx + 8;
        if(y >= h && y < h + dy) {
            if( x >= w && x < w + dx) {
                cell[j][i] = 0;
            }
        }
                    }
                }
                controller.animationPane.repaint();
                controller.repaint();
            }
        });
        f.pack();
        // 表示枠のサイズを決める。
        f.setSize(new
Dimension((int)frameWidth,
        (int)frameHeight));
        f.setVisible(true);
        // アプリケーションのとき，コマンドラ
イン引数から密度を入力
        ks = Double.parseDouble(argv[0]);
    }
}
```