H26 前期研究報告会 研究報告書『迷路の自動生成』

14　第 4 類　竹内和雅

1.研究概要･経緯

迷路を自動で生成するシミュレータを開発した。複数の迷路生成方法を考案し、それぞれのメリットとデメリットについて検証を行った。この研究を行うに至った経緯は、10 月頃に行われるマイクロマウス大会への参加である。迷路探索シミュレータを自作し、自身の迷路探索のアルゴリズムを検証するための第一段階として、迷路の自動生成に着手した。

この報告書では、主に生成の手順と比較に注目して記述する。シミュレータ自体の情報は、最後に資料として添付する。

2.研究目的

迷路生成の方針としては、以下の三つを考えた。

①ランダムに生成する

②一目で最短経路が分からない難易度

③既存のアルゴリズムを参照しない

迷路生成のアルゴリズムを自身で考案することによって迷路探索のコツを得られると考えたため、あえて既存のアルゴリズムを参照しないことを制約に加えた。

3.研究方法

主に 4 つの迷路生成方法を試行した。

①棒倒し法

②小部屋法

③規則沿い法

④歩数マップ確認法

生成した迷路は以下のように分類できる。

①②…マス自体が壁として作用する

③④…壁でマスを区切る

4.研究内容

4.1 各種生成方法の手順

4.1.1 棒倒し法

①n×n 個のマスを用意する。(n は 5 以上の奇数)
左上の座標を(0,0)とし、

$0 \leqq x \leqq n-1$
$0 \leqq y \leqq n-1$ の座標を割り振る。

(右下の座標が(n-1,n-1)となる)

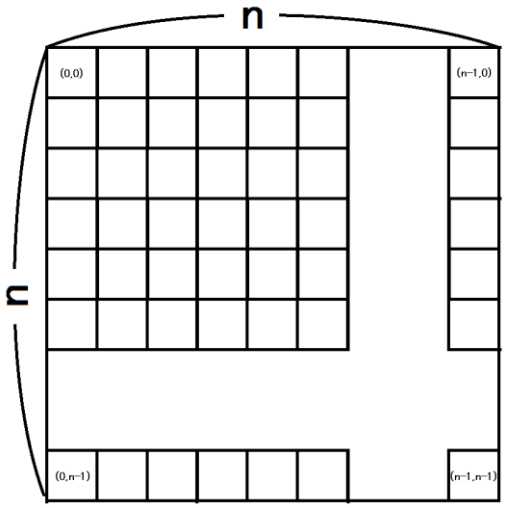


Figure 1　マス目の設定

②最外枠を壁にする。

P_1={(x,y) | (0,k),(n-1,k),(k,0),(k,n-1) k=0,1,…n-1}

③壁から数えて縦横ともに奇数番目に位置するマスに柱を設置する。

$P_2=\{(2i,2j) \mid i=1,\cdots\frac{n-3}{2} \quad j=1,\cdots\frac{n-3}{2}\}$

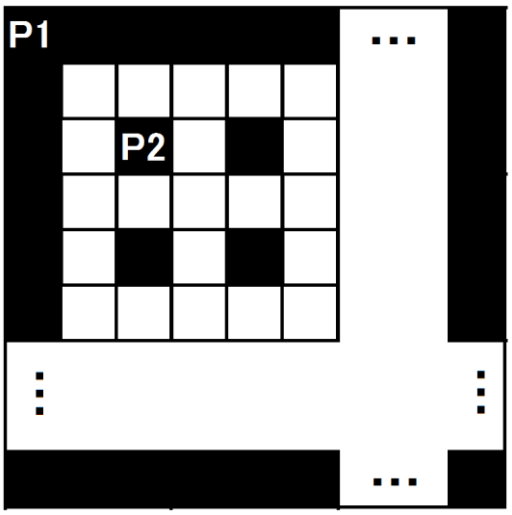


Figure 2　最外枠･柱の設置

④③で作った柱を上下左右いずれかの方向に一マス分倒す。

$P_3^1=\{(2i+1,2j) \mid i=1,\cdots\frac{n-3}{2} \quad j=1,\cdots\frac{n-3}{2}\}$

$P_3^2=\{(2i-1,2j) \mid i=1,\cdots\frac{n-3}{2} \quad j=1,\cdots\frac{n-3}{2}\}$

$P_3^3=\{(2i,2j+1) \mid i=1,\cdots\frac{n-3}{2} \quad j=1,\cdots\frac{n-3}{2}\}$

$P_3^4=\{(2i,2j-1) \mid i=1,\cdots\frac{n-3}{2} \quad j=1,\cdots\frac{n-3}{2}\}$

のいずれかをランダムに壁にする。

(P_3^1: P_3^2: P_3^3: P_3^4=1:1:1:1)

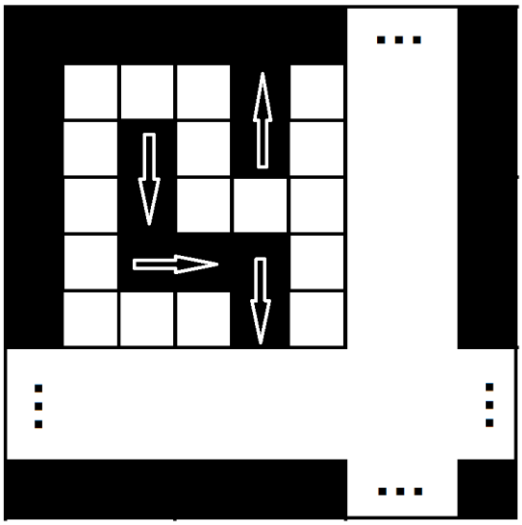


Figure 3　棒倒し

⑥ゴールを設置して完成

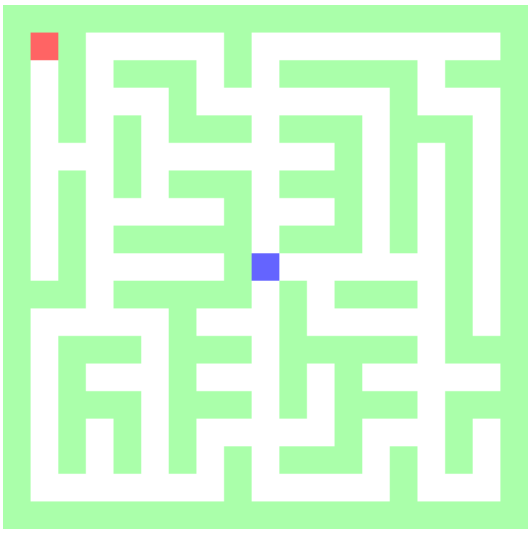


Figure 4　棒倒し法　完成

実際にはゴール座標を中心に置くために、ゴール周辺の壁を取り去る動作を入れている。

4.1.2 小部屋法

①棒倒し法で迷路を複数生成する

②生成した迷路を隣接して設置する

③隣接した各迷路の壁を一か所取り去り、部屋をつなげる

④ゴールを設置する

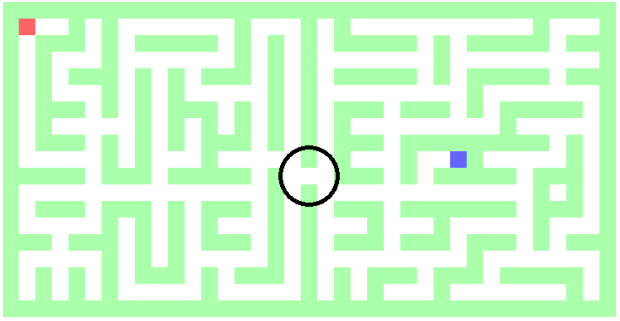


Figure5　小部屋法完成

丸で囲んだ部分が部屋をつなげた部分である。

4.1.3 規則沿い法

①n×n 個のピンを用意する(n は 4 以上の偶数)

各ピンには(x,y)という座標と、(wall_x,wall_y)という壁の有無を示す属性が含まれている。

wall_x=true のとき水平方向の壁が存在し、wall_x=false のときは存在しないとする。

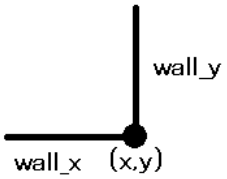


Figure6　規則沿い法　ピン

②ピンを下図のように配置する

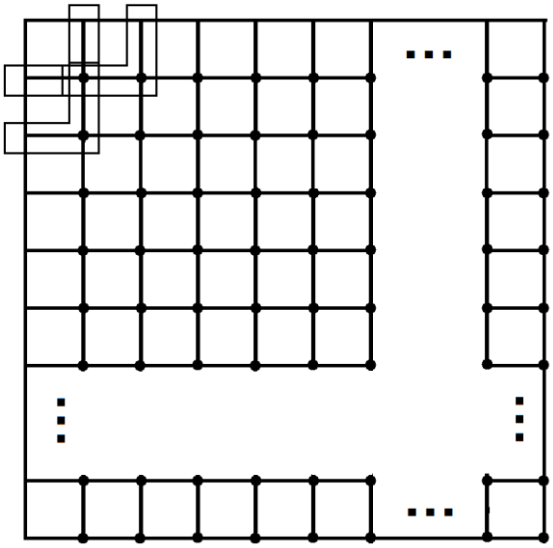


Figure 7　規則沿い法　ピン配置

ここで、ピンの座標は左上から(0,0)、右上が(n-1,0)、左下が(0,n-1)、右下が(n-1,n-1)とする。

③右壁面と下壁面に壁を設置する

P_1={(n-1,k)|k=0,1,…n-1} における wall_y

P_1'={(k,n-1)|k=0,1,…n-1} における wall_x

=true

④L 字という規則に沿いながら壁をなぞり、一定確率で壁を設置する。

P_2={(i,j)|i=0…,n-2 j=0,…i}における wall_y

P_2'={(i,j)|i=0,…j j=0…,n-2 }における wall_x

=(true：false = 1：1)

ただし、L 字のすべてが壁になってしまった場合はもう一度やり直す。

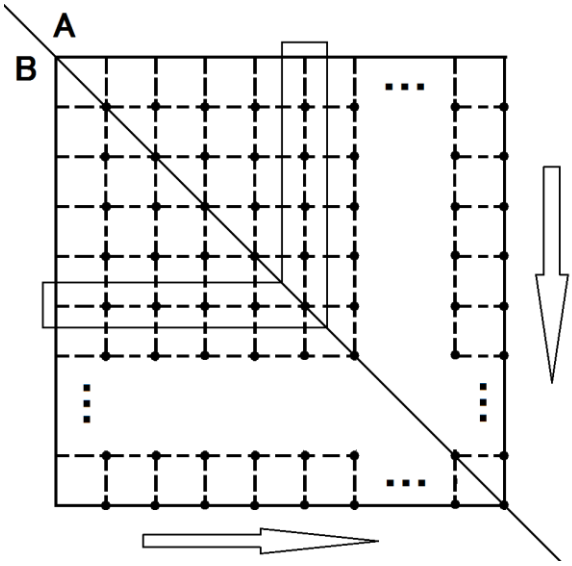


Figure 8　規則沿い法　L 字規則

⑤ランダムにピンを選択する。領域 A に含まれる場合は wall_x=true、領域 B に含まれる場合には wall_y=true とする。これを数回繰り返す。

P_3={(x,y)|x>y, x=random,y=random}における wall_x=true

P_3'={(x,y)|x<y, x=random,y=random}における wall_y=true

ただし、random は 0≦random≦n-1 の乱数

⑥ピンを順番に見ていき、wall_x,wall_y が両方とも false のピンをピックアップする。そのなかで、それに続く壁が設置されていなかった場合のみ壁を増やす。

P_4={(x,y)|(wall_x,wall_y)=(false,false)}

P_4'=(x-1,y)における wall_x=false

⇒における wall_x=true

P_4''=(x,y-1)における wall_y=false

⇒P_4における wall_y=true

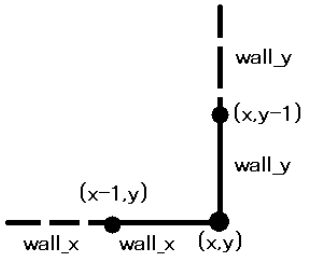


Figure9　規則沿い法　壁追加条件

⑦中心の 4 マスで構成される 2×2 の正方形にゴールを設置して完成

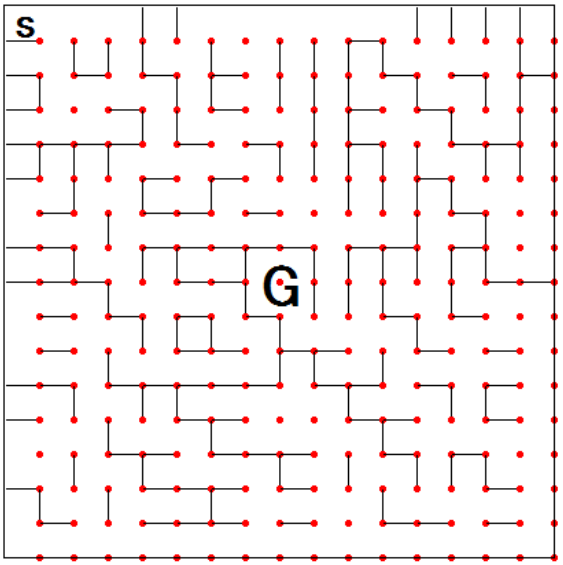


Figure10　規則沿い法　完成

4.1.4 歩数マップ確認法

①横の壁候補の集合である row(r,c)を n-1 行 n 列分と縦の壁候補の集合である column(r,c)を n 行 n-1 列分用意する。ただし n は 4 以上の偶数である。r は何行目かを表し、c は何列目かを表している。(これまでの座標である x,y とは縦横の関係が逆に対応する)

row(r,c)=true のとき、横の壁の集合における r 行 c 列目の壁が存在するということである。

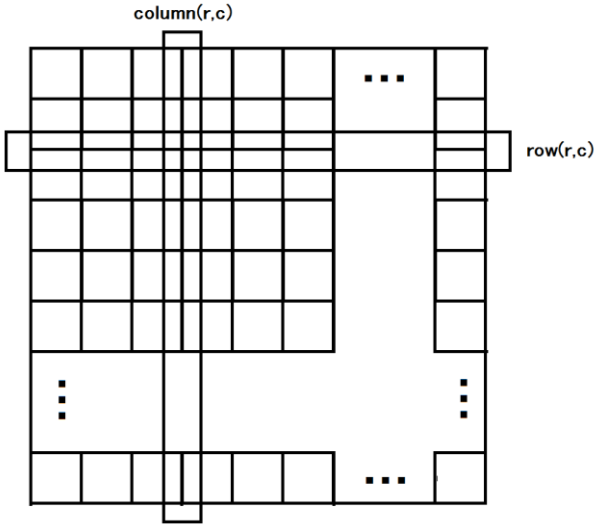


Figure 11　歩数マップ確認法　壁配置

⑦中心の 4 マスで構成される 2×2 の正方形にゴールを設置する。

②ゴールの正方形よりも外側にある、同中心の正方形という規則に沿って、一定確率で壁を設置する。

row($\frac{n}{2}$-i-2,$\frac{n}{2}$-i+j-2)=(true：false = 1：1)

row ($\frac{n}{2}$+i , $\frac{n}{2}$-i+j-2)= (true：false = 1：1)

column($\frac{n}{2}$-i+j-2, $\frac{n}{2}$-i-2)= (true：false = 1：1)

column($\frac{n}{2}$-i+j-2, $\frac{n}{2}$+i)= (true：false = 1：1)

(i=1,2,…$\frac{n}{2}$-2　j=2i+2)

i はゴールから数えて何番目の殻になるか、j は殻の中で何番目の要素かということを表している。

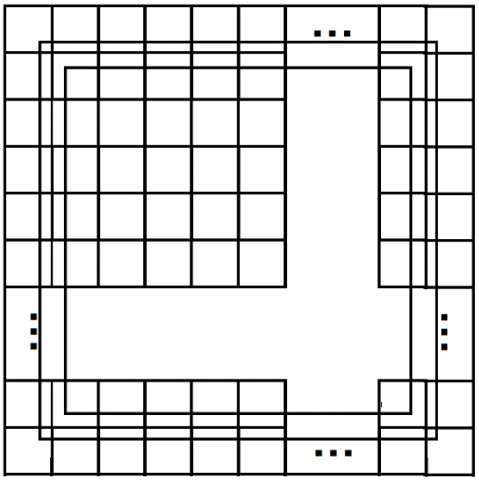


Figure 12　歩数マップ確認法　正方形規則

③壁の設置できる部分(row と column すべて)の中から一つランダムに選択し、false だった場合 true にする。

その後歩数マップ(スタート地点からそのマスに行くために必要な歩数を記したマップ)を作成し、歩数が入らない部分、つまり侵入不可能なマスが存在したら、選択した壁を false に戻す。

④③を複数回繰り返す

⑤ゴール内部にできてしまった壁を false にし、ゴールの入り口を入口候補の中から歩数マップの分布が大きい方向に面する場所に移して完成

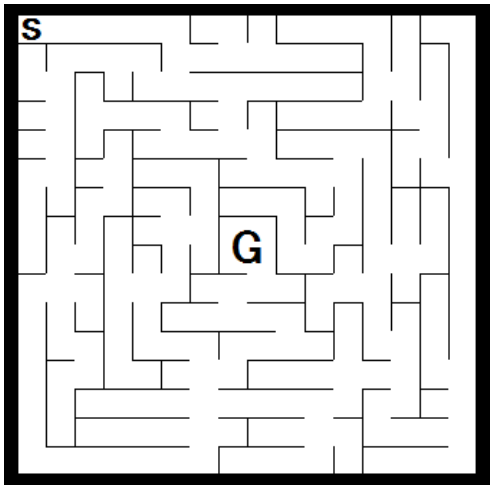


Figure13　歩数マップ確認法　完成

4.2. 生成の性能の比較

上記の方法で生成できる迷路を比較する。

比較するべき項目としては、

①最短経路の長さ

②生成の精度(必ず解ける迷路が生成できるか)

③生成にかかる処理時間

主にこの 3 つである。

最短経路の長さは、複数回生成させた迷路の最短経路の平均をとる。

生成の精度は、複数回生成させた結果からゴールまで到達できる迷路が出力された割合を求める。

生成にかかる時間は、処理にかかった時間をプログラム上に表示させ、その平均をとる。

この二点に加え、それぞれの方法によるメリット･デメリットや視覚的な特徴なども交えながら、総合的に判断する。

なお、PC のスペックによって処理時間は変わるが、大小関係は変わらないと判断する。

5.比較結果

比較結果を下表にまとめる。なお、小部屋法は棒倒し法の延長に存在するため今回の比較からは外した。

Table1　各生成方法の比較結果

	棒倒し法 19×19	規則沿い法 16×16	歩数マップ確認法 16×16
最短経路	18.7	24.5	40.3
生成の精度(%)	100	63	100
処理時間(ms)	0.2	78.6	1331.5
形状の特徴	スタート地点からゴールまでの経路が複数存在する。	ランダム要素が高く、全体としては複雑に見えるが経路自体は短め。	スタート地点からゴールまでの経路がほぼ一通りになる。
メリット	・生成の精度が良い ・単純な手順で作れるため、処理時間が短く実装も簡単	・複雑化した結果、難易度が上がった。	・生成の精度が良い。 ・最短経路が長くなり、難易度が上がった
デメリット	・単純な迷路になってしまい、最短経路が短い	・生成の精度が悪い	・生成に時間がかかる

この結果は、それぞれの生成方法で 50 回生成して、平均をとったものである。この結果から、どのような生成方法が良いか考察する。

6.考察

まずは最短経路の長さに注目する。これは(棒倒し法 ＜ 規則沿い法 ＜ 歩数マップ生成法)という結果になった。歩数マップ生成法の最短経路は他の二倍近くの長さがある。必ず解け、かつ壁が増えるということは、最短経路の増加に直結する。最短経路を増加させるためには、壁を数多く配置すればよいのである。ただ、規則的に置いただけでは単調なものになってしまうため、ランダム要素を用いて複雑化するのである。

それに伴って注目すべき点は生成の精度である。棒倒し法と歩数マップ確認法では 100%の確率で解くことが出来る迷路を生成できたが、規則沿い法では 37%の確率で失敗した。これは、規則沿い方法では閉じてしまう要素が多くなっているからである。L 字の規則に沿って壁を配置した後にランダムに壁が配置されるが、密集している部分に配置されると閉じた空間を作ってしまう可能性が高い。特に次図のような状態に規則配置されると閉じてしまう可能性が高まる。

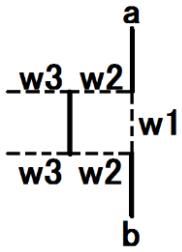


Figure14 規則沿い法の弱点

このように 3 つの壁が配置された場合、a-b 間がつながってしまう可能性が高い。まず、ランダムに選ばれた壁が w1 だった場合、a-b 間はつながる。それに加えて、w3 が配置されていなければ w2 は必ず配置される。w2 が配置されると a-b 間は繋がる。このように繋がった部分が多くなりすぎると、循環して閉じてしまう可能性が高くなる。この他にも閉じてしまう要素となりうる壁の配置が多いため、規則沿い法では生成の精度が悪くなっている。これを防ぐためには、

①閉じた空間を作りにくいアルゴリズムで生成する…棒倒し法

②閉じた空間を検知して、壁を取り除く…歩数マップ確認法

といったような解決策をとる必要がある。

ここまで見ると歩数マップ確認法が最良なように見えるが、処理時間という面において歩数マップ確認法は弱いと言える。壁を置くたびに歩数マップを作って確認するという処理を挟むため、計算量が格段に増える。計算量を抑えるために、壁を置く場所を効果的に選ぶなどの工夫が必要となる。

7.まとめ･今後の展望

難易度が高いが必ず解ける迷路を生成するためには、壁の置き方を工夫するか、必ず解けるか検証して補正する機能が必要となる。今回試した中では、歩数マップ確認法が最も理想に近い迷路を生成できるという結論に至った。だが、処理時間が長いという面を克服するための何らかの方策が必要である。

今後展望としては、処理時間を少なくする工夫をしたい。特に、壁を設置して歩数マップを作成するとき、手数を減らせるような壁の選択方法を考えたいと思う。また、迷路シミュレータの次のステップとして、探索するアルゴリズムを考案したい。

8.資料

8.1　現在の迷路シミュレータの機能

プログラム言語はC++を用い、DXライブラリを適用することで描画処理を行っている。

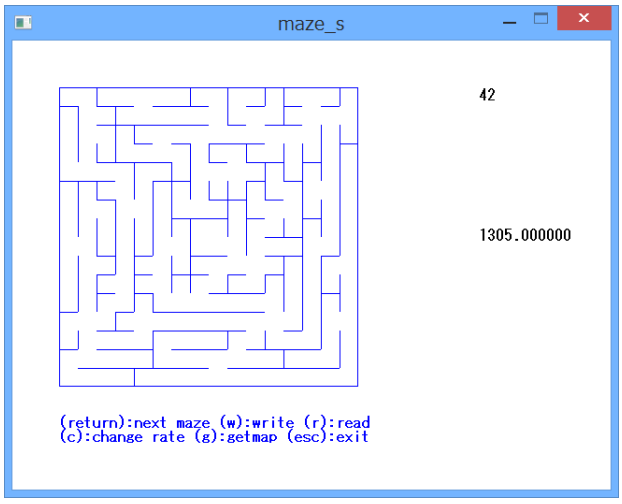


Figure15 迷路シミュレータ

①　迷路生成

上述した方法で迷路を自動で生成する。

②　歩数マップ表示

スタート座標から、全てのマスに対する歩数マップを表示する。歩数マップを作成するアルゴリズムは以下のとおりである。

Ⅰそれぞれの迷路のマス(x,y)に対して対応する数値マス map(x,y)と、数値確定フラグ f_m(x,y)を用意する。また、探索候補を格納するバッファを用意する。

Ⅱスタート地点である(0,0)に対して、map(0,0)=0,f_m(0,0)=true とする。

Ⅲ数値が確定しているマスに隣接し、進行可能な不確定マスをバッファに格納する。

Ⅳバッファ内の不確定マスに注目し、隣接する確定マスの最小値＋1 をそのマスの数値として保持する。

Ⅴ数値が保持されているマスを確定させる。

ⅥⅢ-Ⅴを繰り返して、バッファに格納されるマスが 0 個になったら終了する。

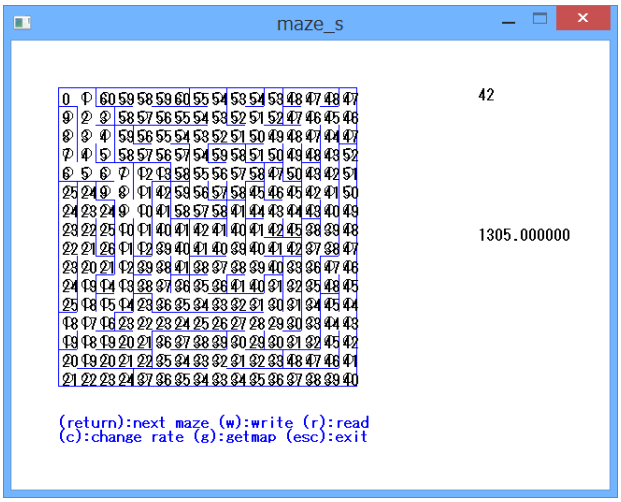


Figure16 歩数マップ表示

③最短経路表示

ゴール座標から、歩数マップが自分より 1 少ないマスをたどっていくとスタート地点まで戻ることが出来る。その経路が最短経路となるため、その経路を表示する。

④データ書き出し読み込み

生成した壁の状態を text ファイルに出力して、状態を保存することが出来る。その text ファイルを呼び出すことによって、状態を復帰することが出来る。

⑤規則沿いで置く壁の確率の変更

正方形の規則に沿って置かれる確率を変更することが出来る。

高確率…正方形の特徴が濃く見える。迷路が直線的になる。

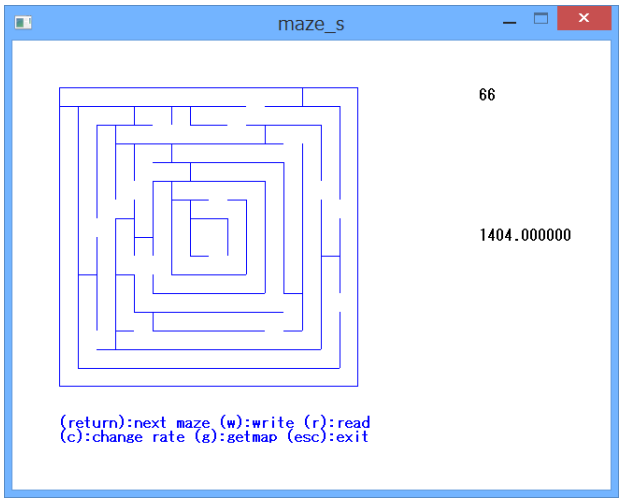


Figure17 規則沿い 80%

低確率…規則性が見えない。曲がり角が多くなる。

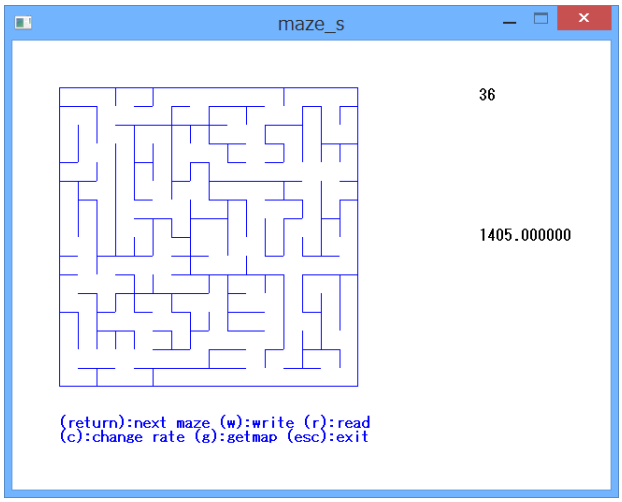
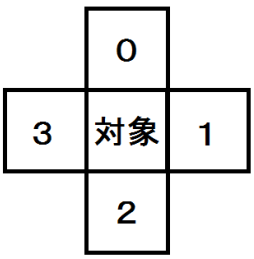


Figure18 規則沿い 20%

現在はここまで完成しており、今後自分で壁を調節するエディットモード、探索モードなどを追加する予定である。

8.2　進行可能という情報をどう渡すか

迷路において、上下左右に進行可能であるか判断する必要があるが、何度も壁を確認することは処理を無駄に増やすことになる。そこで、2 進数を用いることで処理を短縮した。



0000 全方向進行不可
0001 0方向に進行可能
0010 1方向に進行可能
0100 2方向に進行可能
1000 3方向に進行可能

Figure19 方向

上記図のように方向に数字を割り振った。4 ケタの二進数を用意し、進行可能な方向に対応する桁を 1 にする。例えば上と下に進行可能な場合は、0101 となる。そしてその二進数を一つの情報として渡すのである。これを使うときにはシフト演算子を用いてアンド演算を行い、結果が 1 だった場合その方向に進行できると判断することが出来る。具体的に言うと、

1<<d(d=方向に対応する数字)

とアンド演算を行い、結果によって判断する。これによって、情報を一つ渡すだけで 4 つの方向について知ることが出来る。