作品名　　掃除の効率化

所属　　広島大学附属高等学校

３年　長見　啓介　　３年　植村 葵　　３年　岡﨑　文音　　３年　藤川　英

１．研究の要約

本研究の目的は，毎日のように教室で行われる掃除を効率的に進める方法について考察し，数理モデルを用いたプロセッシングでのシミュレーションを通して最適な解を見出すことである。考察にあたっては掃除効率にかかわる要因として「箒と机運びの人数」と「掃除ルート」を設定する。そして，それらのパラメータを変化させながらコンピューターを用いてシミュレーションを行い，その結果と実際の掃除の場面とを比較しながら，効率的な掃除方法について分析を行った。その結果，最も効率的に清掃を進めることができる，箒係と机係の人数比が明らかとなった。今後は，より現実的な場面に近づけて考察を行うために，いくつかの要因を加えてシミュレーションを重ね，学校生活の一部となっている清掃活動のさらなる効率化に貢献していきたいと考えている。

２．研究の動機と目的

塾，部活，生徒会の仕事がありただでさえ時間がない放課後，私たちは限られた時間の中で掃除を行わなければならない。そこで私たちは掃除時間を短縮して少しでも負担を減らすことができないかと思い，この研究を始めた。

本研究の目的は，コンピューターでのシミュレーションによる考察から，一般的なホームルーム教室での最も効率的な掃除の方法を提案することである。そこで，掃除における役割の人数比および掃除ルートをパラメータとして，箒で床を掃く人と机を運ぶ人の動きを表す数理モデルを考え，プロセッシング(Processing)というプログラミングプラットフォームを用いてシミュレーションを行う。ここでの「役割の人数比」とは，何人が机を運び，何人が箒で床を掃くかという人数比のことである。

モデル化にあたってはセルオートマトン（以下CA）モデルを用いる。CAモデルは，マス目上を駒が移動するモデルであり，柳澤・西成（2012）は，駒の動きに簡単なルールを設定するだけで，車の渋滞が後方に進む現象や，狭い出口に人が殺到して出口が詰まる現象などをシミュレーションで再現できることから，現象の本質を見抜くのに有効であると述べている［1］。本研究においてもこの手法を利用して考察を行う。

本研究では教室をマス目に区切り，生徒を駒として CA モデルを用いるが，駒の妥当な動き方（ルール）を明らかにする必要がある。例えば，箒で床を掃く人はどのように動くと考えるのが妥当かということである。このことから,本研究を次のような流れで進めていく。

(1)　妥当なルールを実験から明らかにし CA モデルに導入する。

(2)　シミュレーションによって最も効率的な役割の人数比を明らかにする。

３．方法

箒で床を掃く人を「箒係」，机を運ぶ人を「机係」と呼ぶこととする。本研究では，『教室掃除における箒係と机係の人数に関して，掃除時間を最短にするための最適な人数比が存在する』という仮説のもとで実験，考察を行った。

1）予備調査

予備調査では，実際に本校の生徒たちが教室掃除をしている様子を観察し，撮影を行った。この予備調査は，CA モデルにおける駒の動き方のルールを明らかにすることを目的として実施している。

撮影は，教室を掃除する生徒に掃除の効率化という目的を伝えずに行い，箒係と机係は掃除の途中で役割を変更しないようにした。なお，箒係と机係以外の係の生徒は調査対象から除くこととしている。予備調査を 6 回行

い，調査結果は **表 1** のようになった。

表 1　掃除の係の人数と時間

	箒係(人)	机係(人)	時間
1 回目	3	2	7 分 0 4 秒
2 回目	3	2	5 分 5 1 秒
3 回目	2	3	4 分 5 5 秒
4 回目	2	3	4 分 1 8 秒
5 回目	2	3	5 分 1 0 秒
6 回目	2	3	6 分 0 0 秒

この観察から以下のことが明らかとなった。

・教室掃除の人数については第 2 学年の 5 クラスのうち多くのクラスが 5 人だった。（Ⅱ年 5 組のみ 10 人だったが，2 か所の掃除場所の合計人数だったため，1 つの掃除場所には 5 人と考えた。）

・生徒は，掃除中にほとんど無意識に掃除ルートを決定している。よって，次の行動を決定するには，いくつかの条件がある。

・掃き掃除については，「一度掃いた場所は掃かない」「自分の近くの場所から掃く」「教室の前方から掃く」といった傾向が見られる。また，「箒係の2人が近づいたときに折り返す」「掃き残しがあった場合戻って掃きなおす」といったことも観測された。

2）本調査

本調査では，CA モデルを用いて数理モデルを作成し，シミュレーションを行った（プログラミング環境としてプロセッシングを使用)。シミュレーションにおいては，教室を正方形の 13×15 のマスに分割し，そのマスに箒を掃く人や机を駒として設定する。そして，モデルの作成にあたっては，まず予備調査の結果から掃除をする生徒の行動ルールを定義した。行動ルールとは，生徒が掃除中にほとんど無意識に決定している，掃除ルートの決定条件をルール化したものである。

1．箒係の行動ルール

箒係の行動ルールとして以下の 3 つを設定する。

① 一度はいたマスは掃かない。

② 自分の近くのマスから掃く。

③ 教室の前方から掃く。

そして私たちは箒係がこれらの行動ルールに基づいて行動するよう，各マス目にコストを設定した。コストとは，そのマス目への行きにくさを示し，箒係はよりコストが低いマスに移動する。箒係の行動ルールが数理モデル上の箒係のルートに反映されるよう，以下のようにコストを設定した。

・行動ルール①より，一度はいたマスには 100 という高いコストを設定する。

・行動ルール②より，隣同士の距離を 1，斜め同士の距離を$\sqrt{2}$ として，コストをそのマスとの距離に等しい数値に設定する。

・行動ルール③より，列が後方になるごとにコストを 1 ずつ加える。

図 1 は各マスのコスト設定の例を図に表したものである。中央のマス目にあるひし形は箒係を表し，青色に塗りつぶされているのがすでに掃かれたマス目を表す。各マス目に書かれた数字はコストを示し，矢印は箒係が次に移動する方向を表す。

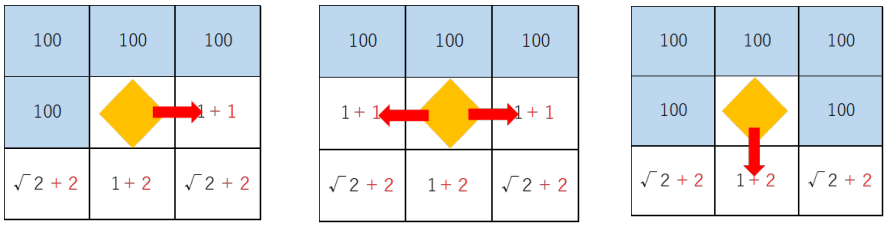

図１　各マスのコスト設定（例）

また，箒係同士の衝突を避けるため zoc というコストを設定した。zoc を箒係がいるマスの周り 8 マスに 2.5 を加える。箒係同士が近づくと周り 3 マスが重なって，その部分の zoc の合計が 5 になるため，2 人の衝突が避けられる。（このコストは常に箒係の周囲に同じ値が加えられるので，個人の掃除ルートに影響を与えない。）図 2 は左から順に，2 人の箒係が接近する様子と，zoc が加えられたマスが重なり，コスト値が高くなる様子を表している。

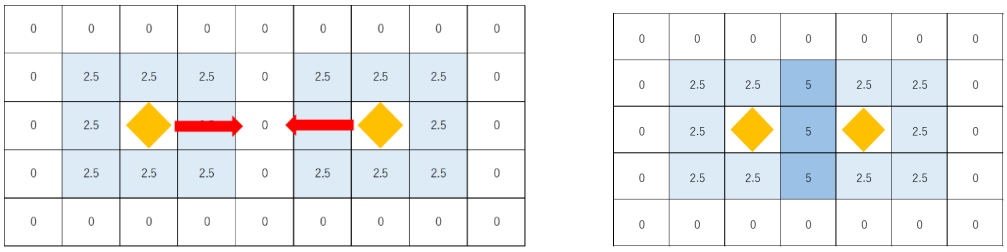

図２　接近とコスト値の上昇

２. 机係の行動ルール

机係の行動ルールは以下の 2 つである。

①机の置かれている前のマスが掃かれたら，その机を運ぶ。

②机係の人数分しか同時に動くことはできない。

数理モデルにおいて，これらのルールは，同時に移動する机の最大数を机係の人数分にすることによって反映される。

４．結果と考察

1）コンピューターによるシミュレーション

以上の行動ルールをもとに作成した数理モデルで，箒係と机係の人数比をパラメーターとして教室掃除のシミュレーションを行った。シミュレーションで設定する数値は，予備調査の結果をもとに設定した。モデル図上では，教室は各マスが正方形になるよう，13×15 マスに分割される。丸が箒係を，オレンジ色の長方形が各机の位置を表し，掃いた場所は青色で示される。教室掃除の人数は 5 人とし，箒係と机係は 1 ステップに 1 マス，縦横斜めのいずれかに移動する。箒で 1 マス床を掃く速さは机が 1 マス運ばれる速さの 2 倍とする。また，教室掃除では，ふつう教室の前方から後方に向かって掃くので，箒係の初期位置は教室の前方に設定した。係の人数比は（箒係：机係）=（1：4），（2：3），（3：2），（4：1）の 4 通りについて実験している。実際のシミュレーションの様子は 図３の通りである。

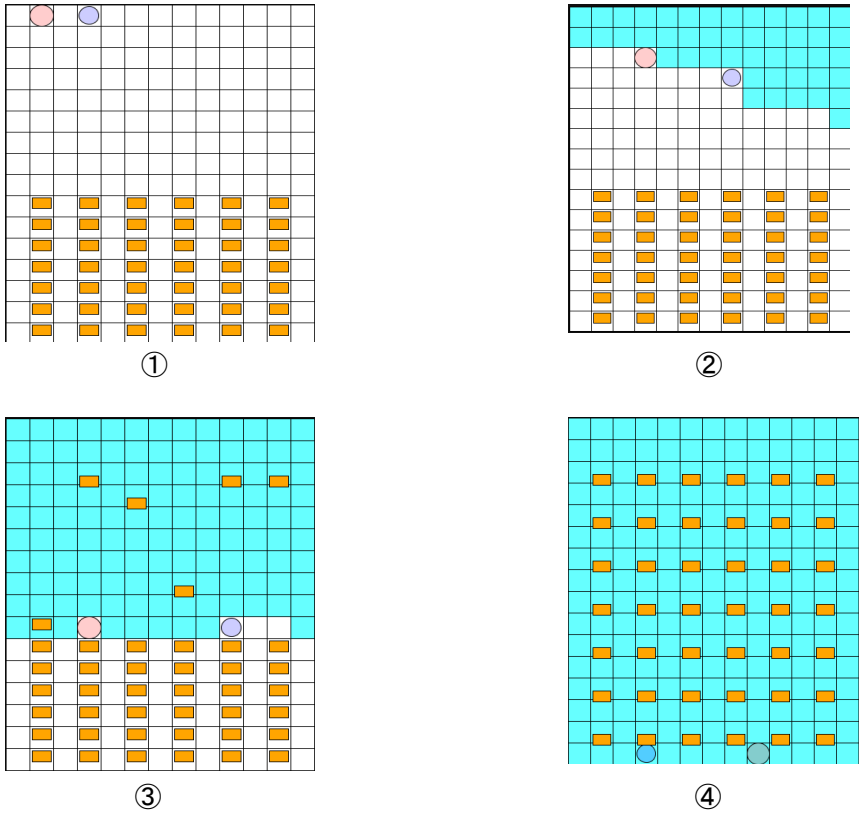

図３　コンピューターによるシミュレーションの様子

これらのシミュレーションをそれぞれの人数比で 100 回ずつ行った結果, 掃除時間は**表２**のようになった（箒係と机係がそれぞれ１マス進むのに必要な時間を１とする)。箒係と机係の人数比が 3 : 2 のときに，掃除時間は最も短くなった。

表２　係の人数比と掃除時間

箒係：机係	平均時間
１：４	４１６
２：３	２６２
３：２	２５７
４：１	３４０

2）考察

実際の教室で観察された，箒係の2人が近づいたときに折り返す様子や，掃き残しがあった場合戻って掃きなおす様子がシミュレーションでも再現されていた。また，シミュレーションを何度も眺めているうちに，掃除は大まかに分けて3つの場面で構成されていることに気づいた。3つの場面とは，最初の「（ア）机が置かれていないスペースを箒係が掃く場面」，次の「（イ）箒係が床を掃きながら同時に机が運ばれていく場面」，最後の「（ウ）ゴミを集める場面」のことである。（ア）は，机係にとっては単なる待ち時間であり，箒係が多いほうが（ア）にかかる時間は短いが，箒係を多くしすぎると（イ）の時間が増える。つまり，（ア）と（イ）の時間のバランスをとることが大切である。

５．結論と今後の課題及び感想

1）結論

今回のシミュレーションでは箒係と机係の人数比が 3 : 2 のとき，掃除時間が最も短くなったが，2 : 3 のときと僅差だったため，本研究では「1 : 4 や 4 : 1 のときよりも 2 : 3，3 : 2 のときが，比較的効率よく掃除をすることができる」ということが言える。この結果は，役割の偏りによって効率が下がったためだと考えられる。また 1 : 4 の方が 4 : 1 より時間がかかったのは，箒が１人しかいないため掃き掃除が机運びに間に合わなかったこ

とが原因であると考えられる。

また，本研究の成果として次の 4 つがあげられる。

①基本的なモデルが作成できたこと

②モデルで実際の掃除の性質を再現できたこと

③モデルから掃除時間が最短になる人数比を明らかにできたこと

④掃除は大まかに分けて 3 つの場面で構成されていること

2）今後の課題

今後の課題としてはモデルの改善が挙げられる。具体的には，次の点に関して改良を行う必要がある。

- 今回のモデルでは箒係が，机が運ばれるのを待たずに掃いていたため，人数比による差が出にくかったこと
- 実際の掃除で多く見られる，掃除中の役割分担を再現すること（例えば，最初は全員で掃き掃除を行い，そこから役割を分担するという掃除方法など）
- ゴミ集めの場面を入れること

より現実的な場面に近づけて考察を行うために，いくつかの要因を加えてシミュレーションを重ね，学校生活の一部となっている清掃活動のさらなる効率化に貢献していきたいと考えている。また，1）の掃除の 3 つの場面の(ア),(イ)のそれぞれにかかる時間を明らかにしたり，3：2 と 2：3 ではどちらのときが最適になるのかということに関しても，さらに詳細に分析し理由も解析していきたい。

3）感想

私たちの研究部ループは，全員プログラミング未経験者で，研究を始めたときは基礎中の基礎から勉強しました。そのため，研究中は初めて知ることばかりで何度も壁にぶつかりました。プログラミングが行きづまっていたときに，偶然学校に講義に来られていた先生にアドバイスをいただいて一気に道が開けたこともありました。研究して感じたのは，人は私たちが思っているよりも臨機応変で複雑に行動しているということです。人の行動をシミュレーションすることの面白さと複雑さは，この臨機応変さにあると感じました。まだまだ課題もある私たちですが，自分たちなりに 1 つのモデルを作り，1 つの結論を出せたことにとても達成感を感じています。このメンバーで研究ができてよかったです。

６．参考文献

［1］ 柳澤大地，西成活裕，「渋滞学のセルオートマトンモデル」， 日本応用数理学会編集『応用数理』，22 巻 1 号，2012 年，2-14.

付録（資料）

Processing によるシミュレーションのプログラムコード

（以下、プログラムコード）

```
int i, j, s, t, n, g, p, q;
int a=1;
int IMAX =13; /*num of side cell max*/
int JMAX =16; /*num of hight cell max*/
int SMAX = 2; /*num of sweeper */
int PMAX = 6;
int QMAX = 7;
int AoI_left = 3;
float smallest_pena;
int dx=30;

PVector[] s_location = new PVector[SMAX+1];
PVector[] s_velocity = new PVector[SMAX+1];
color[] s_col = new color[SMAX+1];
float[] diameter = new float[SMAX+1];
int[][] floor = new int[IMAX+2][JMAX+2];
float[] smallest_i = new float[SMAX+1];
int [] smallest_j = new int[SMAX+1];
int [] goto_i = new int[10];
int [] goto_j = new int[10];
int [] lsi = new int[SMAX+1];
int [] lsj = new int[SMAX+1];
int[][]x_D = new int[PMAX+1][QMAX+1];
int[][]y_D = new int[PMAX+1][QMAX+1];
int[][]AoI = new int[PMAX+1][QMAX+1];

float[][] pena = new float[IMAX+2][JMAX+2];
float[][] zoc = new float[IMAX+2][JMAX+2];
float v_zoc=2.5;
float[][] d_o = new float[IMAX+2][JMAX+2];
int check_d = 0;
int check_f = 0;



void setup() {
  size(550, 550);
  //println ((int)(dx*(IMAX+2)),(int)(dx*(JMAX+2)));
  frameRate(10);
```

```
t=0;
//floor,pena initialize
for (i=0; i<=IMAX+1; ++i)
{
   for (j=0; j<=JMAX+1; ++j)
   {
      floor[i][j]=0;
   }
}
for (i=0; i<=IMAX+1; ++i)
{
   pena[i][0] = 200;
   pena[i][JMAX+1] = 200;
}
for (j=0; j<=JMAX+1; ++j)
{
   pena[0][j] = 200;
   pena[IMAX+1][j] = 1000;
}

//draw area 1st
for (i=1; i<=IMAX; ++i)
{
   for (j=1; j<=JMAX; ++j)
   {
      fill(255-floor[i][j]*255, 255-floor[i][j]*255);
      rect(dx*i, dx*j, dx, dx);
   }
}

for (s = 0; s < SMAX; s++)
{
   s_location[s] = new PVector(2+s*2, 1);
   s_velocity[s] = new PVector(0, 0);
   s_col[s] = color(s*80, 255-s*80, s*80, 50);
   diameter[s] = random(30-5*s, 30-5*s);
   fill(s_col[s]);
   //println (s_location[s].x*dx);
   ellipse(s_location[s].x*dx+dx/2, s_location[s].y*dx+dx/2, diameter[s], diameter[s]);
}

for (q=1; q<=QMAX; ++q)
```

```
    {
        for (p=1; p<=PMAX; ++p)
        {
            x_D[p][q] = 2*p*dx;
            y_D[p][q] = (q+9)*dx;
            rect( x_D[p][q], y_D[p][q], dx, dx );
        }
    }
}
void draw()
{
    //draw area
    background(15);
    for (i=1; i<=IMAX; ++i)
    {
        for (j=1; j<=JMAX; ++j)
        {
            fill(255-floor[i][j]*255, 255-floor[i][j]*255, 255);
            rect(dx*i, dx*j, dx, dx);
        }
    }

    //get agent lacation to sweep
    for (s = 0; s < SMAX; s++)
    {
        lsi[s]= (int)(s_location[s].x);
        lsj[s]= (int)(s_location[s].y);
    }

    //initalize zoc
    for (i=0; i<=IMAX+1; ++i)
    {
        for (j=0; j<=JMAX+1; ++j)
        {
            zoc [i][j]=0.0;
        }
    }



    //culculate zoc
    for (s = 0; s < SMAX; s++)
    {
```

```
    for (i=lsi[s]-1; i<=lsi[s]+1; i++)
    {
        for (j=lsj[s]-1; j<=lsj[s]+1; j++)
        {
            zoc [i][j]=zoc [i][j]+v_zoc;
        }
    }
}

for (s = 0; s < SMAX; s++)
{
    for (i=lsi[s]-1; i<=lsi[s]+1; i++)
    {
        for (j=lsj[s]-1; j<=lsj[s]+1; j++)
        {
            for (q=1; q<=QMAX; ++q)
            {
                for (p=1; p<=PMAX; ++p)
                {
                    if (floor[i][j]==0 && i== x_D[p][q] && j==y_D[p][q])
                    {
                        d_o[i][j]=150;
                    } else
                    {
                        d_o[i][j]=0;
                    }
                }
            }
        }
    }
}

for (s = 0; s < SMAX; s++)
```

```
{

  //culculate pena
  for (i=lsi[s]-1; i<=lsi[s]+1; i++)
  {
    for (j=lsj[s]-1; j<=lsj[s]+1; j++)
    {
      if (floor[i][j]==1)
      {
        pena[i][j]=100;
      } else
      {
        if (i>0 && i<IMAX+1 && j>0 && j<JMAX+1)
        {
          pena[i][j]=sqrt(pow(lsi[s]-i, 2)+pow(lsj[s]-j, 2))+(j-lsj[s])+zoc [i][j]-v_zoc + d_o[i][j];
          //    println (lsi[s], lsj[s], i, j,pena[i][j], smallest_pena,floor[lsi[s]][lsj[s]]);
        }
      }
    }
  }

  //culculate smallest pena
  smallest_pena = 1000;
  for (i=lsi[s]-1; i<=lsi[s]+1; i++)
  {
    for (j=lsj[s]-1; j<=lsj[s]+1; j++)
    {
      if (pena[i][j] < smallest_pena)
      {
        smallest_pena = pena[i][j];
        smallest_i[s] = i;
        smallest_j[s] = j;
        //println (smallest_pena, lsi[s], lsj[s], i, j, smallest_i[s], smallest_j[s]);
      }
    }
  }
  n=0;
  for (i=lsi[s]-1; i<=lsi[s]+1; i++)
  {
    for (j=lsj[s]-1; j<=lsj[s]+1; j++)
    {
      if (pena[i][j] == smallest_pena)
```

```
        {
          goto_i[n]=i;
          goto_j[n]=j;
          n=n+1;
        }
      }
    }
    g=(int)random(0, n);
    // println (s, n);


    //sweeper moving to
    s_location[s] = new PVector(goto_i[g], goto_j[g]);
    // println (t, s, s_location[s], floor[lsi[s]][lsj[s]]);    //draw sweeper
    fill(s_col[s]);
    ellipse(s_location[s].x*dx+dx/2, s_location[s].y*dx+dx/2, diameter[s], diameter[s]);

    floor[lsi[s]][lsj[s]]=1; /*just sweep!*/
}



fill(255, 165, 0);
//inactive to active
for (q=1; q<=QMAX; ++q)
{
    for (p=1; p<=PMAX; ++p)
    {
      if (floor[ x_D[p][q]/dx][y_D[p][q]/dx-1]==1 && AoI_left >    0 &&    AoI[p][q] == 0 )
      {
        AoI[p][q] = 1;
        AoI_left = AoI_left-1;
      }
      //active to inactive
      if (y_D[p][q]/dx==q*2+1 &&    AoI[p][q] == 1)
      {
        AoI[p][q] = 0;
        AoI_left = AoI_left+1;
      }
    }
}
//draw desk
```

```
for (q=1; q<=QMAX; ++q)
{
   for (p=1; p<=PMAX; ++p)
   {
      if (floor[ x_D[p][q]/dx][y_D[p][q]/dx-1]==1 && AoI[p][q] == 1 && y_D[p][q]/dx!=q*2+1)
      {
         y_D[p][q]-=dx/2;
         rect( x_D[p][q]+dx/10, y_D[p][q]+dx/10, dx/10*8, dx/10*5 );
      } else
      {
         rect( x_D[p][q]+dx/10, y_D[p][q]+dx/10, dx/10*8, dx/10*5 );
      }
   }
}

//println( AoI_left);

check_d = 0;
check_f = 0;

for (q=1; q<=QMAX; ++q)
{
   for (p=1; p<=PMAX; ++p)
   {
      check_d = check_d + y_D[p][q]/dx - (q*2+1);
   }
}
for (i=1; i<=IMAX; ++i)
{
   for (j=1; j<=JMAX; ++j)
   {
      check_f = check_f + floor[i][j] - 1;
   }
}




if (check_d==0 && check_f==0)
{
   t=t+0;
} else
{
```

```
    t=t+1;
  }
  println(t, check_d, check_f);
}



void keyPressed()
{
  if (keyCode == ENTER)
  {
    save("sample" + a + ".png");
    a++;
  }
}
```