セルオートマトン法による
渋滞のシミュレーション

卒業研究レポート

明治大学 総合数理学部 現象数理学科
4 年 2 組 27 番
学籍番号：2610180003
土井 梨沙

2022 年 2 月 28 日

目次

1 はじめに

2 セルオートマトン法

3 用語

2-1 セル

2-2 ステップ

2-3 周期境界条件

2-4 渋滞クラスター

4 Rule184

4-1 Rule184 のルール

4-2 Rule184 のシミュレーション

5 Quick-Start

5-1 Quick-Start のルール

5-2 Quick-Start のシミュレーション

6 Slow-Start

6-1 Slow-Start のルール

6-2 Slow-Start のシミュレーション

7 Two-lane quick start

7-1 Two-lane quick start のルール

7-2 Two-lane quick start のシミュレーション

8 まとめ

9 謝辞

10 参考文献

1 はじめに

日本における交通事故の８割はドライバーに起因していると言われている。最近では、飲酒運転や居眠り運転などが原因で起きた交通事故を報道番組でよく見かける。このような数あるヒューマンエラーの中でも「交通事故の発生は人の感情が関与している」と記載された記事を見かけた。中でも多いのが「怒り」の感情であるという。さらに、その記事には、運転中にこの感情が発生する場面として、「渋滞」で中々前に進まない時が挙げられると記載されていた。つまり、少しでも渋滞の発生を抑えることができれば、交通事故を減らすことができるということである。そこで私は、日本の交通事故の起因に関与する「渋滞」が発生する原因を知るために本研究に取り組んだ。本論文では、コンピューターシミュレーションにより仮想の渋滞を作り、現実に起こっている様々な条件下で、渋滞が起きる道路上での車の動きを再現することを目標とする。

2 セルオートマトン法

この節の内容は、渋滞学においてセルオートマトンモデルによる研究を行っている西成活裕氏著書「渋滞学」[1]を参考に書き進めるものとする。セルオートマトン法とは、セルの集まり(道路)の上を、粒子(車)をルールに従って動かす方法である。本来なら人や車は道の上を「連続的」に動くが、この方法は人や車を「離散的」に動かすモデルを用いている。このようにセルオートマトン法は、複雑な対象を簡素化して考え、近似して表現したいときによく用いられる。このモデルに様々なルールを導入することで、より複雑な現象や限られた条件下におけるシミュレーションを行うことができる。また、セルオートマトン法ではよく０と１だけで世界を表現する。渋滞学の場合、車が存在するセルを１で表し、車が存在しないセルを０として１が動くルールを色々設定することで様々な現象を表すというのがセルオートマトン法の最も重要な考え方である。本研究では１の代わりに、１つ前のステップから現在いるステップまでに位置が進んだ車を表す G(Go)、１つ前のステップから現在いるステップまでに位置が進んでいない車を表す S(Stop)のアルファベットを用いて現象を表すものとする。また、０はセルに車が存在していないことが分かりやすいように空欄で表すものとする。

3 用語

本論文を読むにあたって必要となる用語の意味を記載する。

2-1 セル

セルとは、道路を車１台分の大きさのマスに区切った、その離散的な１つ分のマス目のことである。このセル上を走る車の進行方向はx軸の正方向とする。

2-2 ステップ

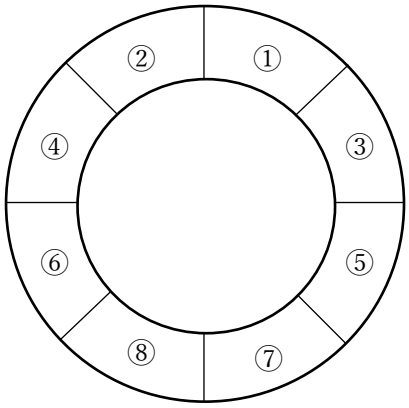
ステップとは、時刻を離散化し自然数にしたその時刻のことである。本論文上では、これを「t」と表すことにする。また、本研究では各シミュレーション 100 ステップ施行する。

2-3 周期境界条件

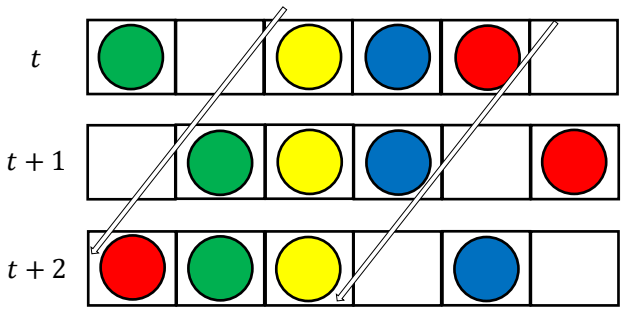
周期境界条件とは、最後のセル(⑧)と最初のセル(①)は接続されているという考え方である。最後のセルにいる車が進行方向へ進む時、最初のセルを前方のセルと考え、最初のセルに進む。(図１参照)

2-4 渋滞クラスター

渋滞クラスターとは、車の数が多くなるとお互いが邪魔になって動くことのできない車の集団が発生し、時間の経過ごとに進行方向とは逆に動いてゆくように見えるその集団のことである。渋滞クラスターの先頭からは渋滞から解放された車が次々と出てゆき、後ろには渋滞に巻き込まれる車が次々に到着する。このように先頭から車が出て、後方から車が入れば、渋滞クラスター全体が後ろに一つ移動したように見える。(図２参照)



【図１】周期境界条件イメージ図



【図２】渋滞クラスターイメージ図

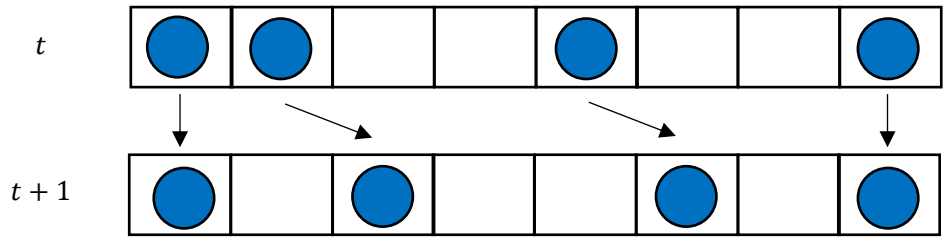
4 Rule184

4-1 Rule184 のルール

１つ目のルールはセルオートマトンモデルの中で最も基本的かつ単純である「Rule184」というルールである。このルールの条件は以下の通りである。

①周期境界条件
②１つ前のセルに車が存在しない場合、次のステップで進む。
③１つ前のセルに車が存在する場合、次のステップで進むことができない。

図３は Rule184 の動き方である。



【図３】Rule184

このルールの下、C 言語でシミュレーションする。

4-2 Rule184 のシミュレーション

以下は Rule184 の C 言語によるシミュレーション・プログラム Rule184.c である。

プログラム Rule184.c

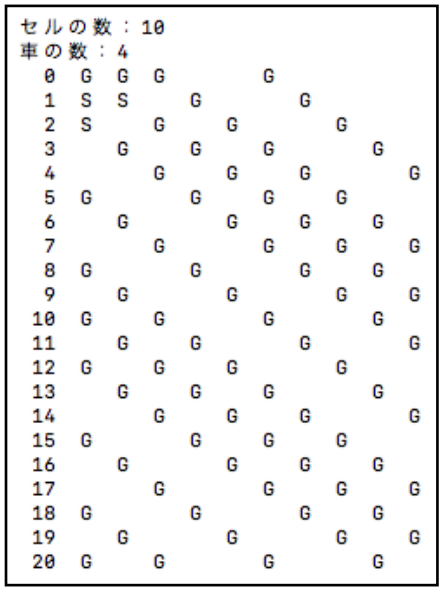
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(int argc, char **argv){
   int C;   //セルの数
   double M;//車の数

   if (argc != 3) {
        printf("セルの数：");
        scanf("%d",&C);
        printf("車の数：");
        scanf("%lf",&M);
   }
   else{
        C = atoi(argv[1]);
        M = atof(argv[2]);
   }
   //C個のセルを持つ配列を用意
   int n = 1;
   int a[C];
   int a2[C];
   char b[4]={' ','G','S','W'};
   // ：車が存在しない
   //G:走っている状態
   //S:止まっている状態
  //W:待っている状態

   //初期化
   for(int i=0; i<C; i++){
        a[i]=0;
        a2[i]=0;
```

```
}

//M台の車をランダムに配置
 int value;
 value=0;
 srand(0);
 while(value<M){
      int r=rand()%C;   //ランダムに車を配置

      if(a[r]==0){      //rのマスが空いていれば
        a[r]=1;        //そのrのマスに1台配置する
        value=value+1;//M台になるまで続ける
      }
 }

 printf("%3d",n-1);
 printf("");

 for(int i=0; i<C; i++){
      printf("%3c",b[a[i]]);
 }
 printf("¥n");

 for(int x=0; x<100; x++){
      //rulr184の条件
      for(int i=C-1; i>=0; i--){
        if(i == C-1){
              //周期境界条件(先頭)
              if(a[C-1] == 1){
                //車が存在するとき
                if(a[0] == 0){
                   //前に車が存在しないとき
                   a2[C-1] = 0;
                   a2[0] = 1;
                }else{
                   //前に車が存在するとき
                   a2[C-1] = 2;
                }
            }else if(a[C-1] == 2){
                //止まっている状態のとき
                if(a[0] == 0){
                    //前に車が存在しないとき
                    a2[C-1] = 0;
                    a2[0] = 1;
                }else{
                    //前に車がいるとき
                    a2[C-1] = 2;
                }
            }
        }else{
            //その他の場所
              if(a[i] == 1){
                //車が存在するとき
                if(a[i+1] == 0){
```

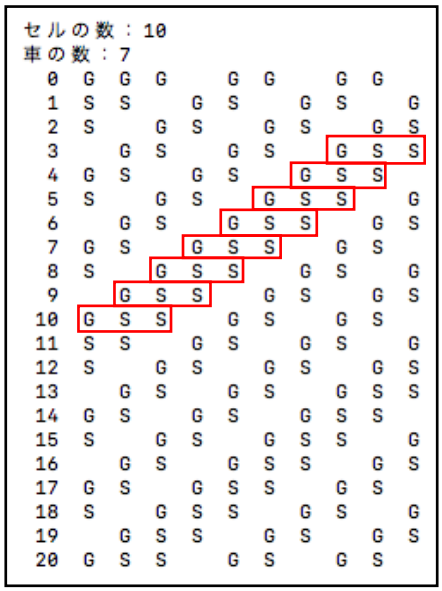
```
                    //前に車が存在しないとき
                    a2[i] = 0;
                    a2[i+1] = 1;
                }else{
                    //前に車が存在するとき
                    a2[i] = 2;
                }
            }else if(a[i] == 2){
                    //止まっている状態のとき
                    if(a[i+1] == 0){
                    //前に車が存在しないとき
                    a2[i] = 0;
                    a2[i+1] = 1;
                }else{
                    //前に車が存在するとき
                    a2[i] = 2;
                }
              }
            }
     }

    for(int i=0; i<C; i++){
        a[i] = a2[i];//次の状態のセルを現在の状態に変える
        a2[i] = 0;//次のターンをリセット
    }
    printf("%3d" ,n++);//ステップ数
    printf("");

    for(int i=0; i<C; i++){
        printf("%3c",b[a[i]]);//車が存在しない時は空欄、車が存在する時はG,Sの車の状態で表す
    }

    printf("¥n");
   }
}
```

以下はプログラム Rule184.c の実行結果である。

Rule184.c の実行結果



```
セルの数：10
車の数：4
 0  G  G  G        G
 1  S  S     G        G
 2  S     G     G        G
 3     G     G     G        G
 4           G     G     G        G
 5  G           G     G     G
 6     G           G     G     G
 7           G           G     G     G
 8  G           G           G     G
 9     G           G           G     G
10  G     G           G           G
11     G     G           G           G
12  G     G     G           G
13     G     G     G           G
14           G     G     G           G
15  G           G     G     G
16     G           G     G     G
17           G           G     G     G
18  G           G           G     G
19     G           G           G     G
20  G     G           G           G
```



```
セルの数：10
車の数：7
 0  G  G  G     G  G     G  G
 1  S  S     G  S     G  S     G
 2  S     G  S     G  S     G  S
 3     G  S     G  S     G  S  S
 4  G  S     G  S     G  S  S
 5  S     G  S     G  S  S     G
 6     G  S     G  S  S     G  S
 7  G  S     G  S  S     G  S
 8  S     G  S  S     G  S     G
 9     G  S  S     G  S     G  S
10  G  S  S     G  S     G  S
11  S  S     G  S     G  S     G
12  S     G  S     G  S     G  S
13     G  S     G  S     G  S  S
14  G  S     G  S     G  S  S
15  S     G  S     G  S  S     G
16     G  S     G  S  S     G  S
17  G  S     G  S  S     G  S
18  S     G  S  S     G  S     G
19     G  S  S     G  S     G  S
20  G  S  S     G  S     G  S
```

ここでは 100 ステップ施行した内の 20 ステップの実行結果を 2 つ提示する。左はセルの数 10、車の数 4 での実行結果、右はセルの数 10、車の数 7 での実行結果である。1 番左の数はステップを表している。また、空欄は車が存在しないことを示す。さらに、冒頭でも述べた通り、「G」は 1 つ前のステップから現在いるステップまでに位置が進んだ車、「S」は 1 つ前のステップから現在いるステップまでに位置が進んでいない車を表している。このように実行結果では、Rule184 以降も車の状態をアルファベットで分けて表示するものとする。左の実行結果を見ると、セルに対して車の数が少ない場合、車の状態は、十分時間が経過すると S がなくなり、G だけになることがわかる。また、右の実行結果を見ると、赤枠で囲った渋滞クラスターが進行方向とは逆に動いてゆくように見えるのを確認することができる。これは、車 1 台が渋滞クラスターから出ると同時に、後ろから 1 台入るという現象が繰り返され起きた現象である。したがって、Rule184 ではこの現象が繰り返され、渋滞クラスター全体が 1 つ後ろに移動したように見えることをシミュレーションにより確認することができた。

5 Quick-start

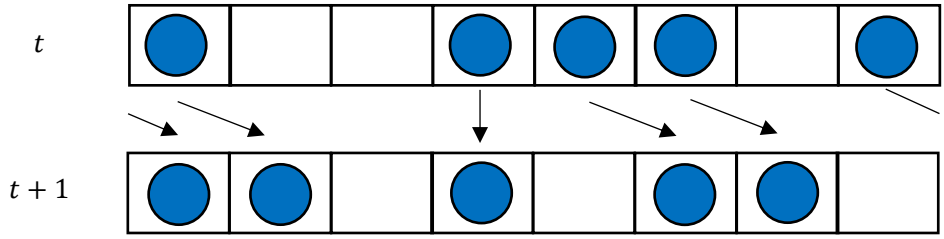
5-1 Quick-Start のルール

2 つ目のルールは 2 つ先のセルの車の動きも見る「Quick-Start」というルールである。このルールの条件は以下の通りである。

①周期境界条件
② 1 つ前のセルに車が存在しない場合、次のステップで進む。

③１つ前のセルに車が存在しているが、２つ前のセルに車が存在しない場合、１つ前の車が進むことを見越して次のステップで進む。
④１つ前のセルに車が存在しており、２つ前のセルにも車が存在している場合、次のステップで進むことはできない。

図４は Quick-Start の動き方である。



【図 4】Quick-Start

このルールの下、C 言語でシミュレーションする。

5-2 Quick-Start のシミュレーション

以下は Quick-Start の C 言語によるシミュレーション・プログラム Quick-Start.c である。

プログラム Quick-Start.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(int argc, char **argv){
    int C;   //セルの数
    double M;//車の数

    if (argc != 3) {
        printf("セルの数：");
        scanf("%d",&C);
        printf("車の数：");
        scanf("%lf",&M);
    }
    else{
        C = atoi(argv[1]);
        M = atof(argv[2]);
    }
    //C個のセルを持つ配列を用意
    int n = 1;
    int a[C];
    int a2[C];
    char b[4]={' ','G','S','W'};
    // :車が存在しない
```

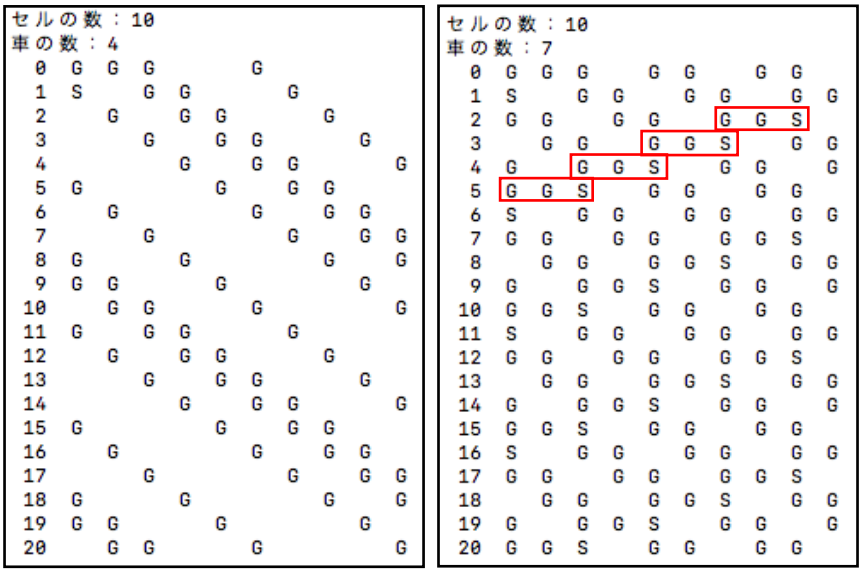
```
//G:走っている状態
//S:止まっている状態
//W:待っている状態

//初期化
for(int i=0; i<C; i++){
    a[i]=0;
    a2[i]=0;
}
//M台の車をランダムに配置
int value;
value=0;
srand(0);
while(value<M){
    int r=rand()%C;   //ランダムに車を配置

    if(a[r]==0){      //rのマスが空いていれば
        a[r]=1;       //そのrのマスに1台配置する
        value=value+1;//M台になるまで続ける
    }
}

printf("%3d ",n-1);
printf("");

for(int i=0; i<C; i++){
    printf("%2c ",b[a[i]]);
}
printf("¥n");

for(int x=0; x<100; x++){
    //Quick-startの条件
    for(int i=C-1; i>=0; i--){
        if(i==C-1){
            //周期境界条件(先頭)
            if(a[C-1] == 1){
                //車が存在するとき
                if(a[0] == 0 || (a[0] == 1 && a[1] == 0) || (a[0] == 2 && a[1] == 0)){
                    //前に車が存在しないとき or 前に車が存在しているが、2個前に車が存在しないとき
                    a2[C-1] = 0;
                    a2[0] = 1;
                }else{
                    //前に車が存在しているかつ、2個前にも車が存在しているとき
                    a2[C-1] = 2;
                }
            }else if(a[C-1] == 2){
                //止まっている状態のとき
                if(a[0] == 0 || (a[0] == 1 && a[1] == 0) || (a[0] == 2 && a[1] == 0)){
                    //前に車が存在しないとき or 前に車が存在しているが、2個前に車が存在しないとき
                    a2[C-1] = 0;
                    a2[0] = 1;
                }else{
                    //前に車が存在しているかつ、2個前にも車が存在しているとき
                    a2[C-1] = 2;
```

```
        }
    }
}else if(i==C-2){
    //周期境界条件（前から2つ目の場所）
    if(a[C-2] == 1){
        //車が存在するとき
        if(a[C-1] == 0 || (a[C-1] == 1 && a[0] == 0) || (a[C-1] == 2 && a[0] == 0)){
            //前に車が存在しないとき or 前に車が存在しているが、2個前に車が存在しないとき
            a2[C-2] = 0;
            a2[C-1] = 1;
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[C-2] = 2;
        }
    }else if(a[C-2] == 2){
        //止まっている状態のとき
        if(a[C-1] == 0 || (a[C-1] == 1 && a[0] == 0) || (a[C-1] == 2 && a[0] == 0)){
            //前に車が存在しないとき or 前に車が存在しているが、2個前に車が存在しないとき
            a2[C-2] = 0;
            a2[C-1] = 1;
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[C-2] = 2;
        }
    }
}else if(i == 0){
    //最後尾
    if(a[0] == 1){
        //車が存在するとき
        if(a[1] == 0 || (a[1] == 1 && a[2] == 0) || (a[1] == 2 && a[2] == 0)){
            //前に車が存在しないとき or 前に車が存在しているが、2個前に車が存在しないとき
            if(a2[0] == 1){
                //先頭の車との重複防止
                a2[1] = 1;
            }else{
                a2[0] = 0;
                a2[1] = 1;
            }
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[0] = 2;
        }
    }else if(a[0] == 2){
        //止まっている状態のとき
        if(a[1] == 0 || (a[1] == 1 && a[2] == 0) || (a[1] == 2 && a[2] == 0)){
            //前に車が存在しないとき or 前に車が存在しているが、2個前に車が存在しないとき
            if(a2[0] == 1){
                //先頭の車との重複防止
                a2[1] = 1;
            }else{
                a2[0] = 0;
                a2[1] = 1;
            }
        }else{
```

```
                    //前に車が存在しているかつ、2個前にも車が存在しているとき
                    a2[0] = 2;
                }
            }
        }else{
            //その他の場所
            if(a[i] == 1){
                //車が存在するとき
                if(a[i+1] == 0 || (a[i+1] == 1 && a[i+2] == 0) || (a[i+1] == 2 && a[i+2] == 0)){
                    //前に車が存在しないとき or 前に車が存在しているが、2個前に車が存在しないとき
                    a2[i] = 0;
                    a2[i+1] = 1;
                }else{
                    //前に車が存在しているかつ、2個前にも車が存在しているとき
                    a2[i] = 2;
                }
            }else if(a[i] == 2){
                //止まっている状態のとき
                if(a[i+1] == 0 || (a[i+1] == 1 && a[i+2] == 0) || (a[i+1] == 2 && a[i+2] == 0)){
                    //前に車が存在しないとき or 前に車が存在しているが、2個前に車が存在しないとき
                    a2[i] = 0;
                    a2[i+1] = 1;
                }else{
                    //前に車が存在しているかつ、2個前にも車が存在しているとき
                    a2[i] = 2;
                }
            }
        }

        for(int i=0; i<C; i++){
        a[i] = a2[i];//次の状態のセルを現在の状態に変える
        a2[i] = 0;//次のターンをリセット
        }

    printf("%3d" , n++);//ステップ数
    printf("");

    for(int i=0; i<C; i++){
        printf("%3c" , b[a[i]]);//車が存在しない時は空欄、車が存在する時はG,Sの車の状態で表す
    }
        printf("¥n");
    }
}
```

以下はプログラム Quick-Start.c の実行結果である。

Quick-Start.c の実行結果



```
セルの数：10
車の数：4
  0  G  G  G        G
  1  S     G  G        G
  2     G     G  G           G
  3           G     G  G           G
  4              G     G  G           G
  5  G              G     G  G
  6     G              G     G  G
  7        G              G     G  G
  8  G           G              G     G
  9  G  G           G              G
 10     G  G           G              G
 11  G     G  G           G
 12     G     G  G           G
 13        G     G  G           G
 14           G     G  G           G
 15  G              G     G  G
 16     G              G     G  G
 17        G              G     G  G
 18  G           G              G     G
 19  G  G           G              G
 20     G  G           G              G
```

```
セルの数：10
車の数：7
  0  G  G  G     G  G     G  G
  1  S     G  G     G  G     G  G
  2  G  G     G  G     G  G  S
  3     G  G        G  G  S     G  G
  4  G     G  G  S     G  G     G
  5  G  G  S     G  G     G  G
  6  S     G  G     G  G     G  G
  7  G  G     G  G     G  G  S
  8     G  G     G  G  S     G  G
  9  G     G  G  S     G  G     G
 10  G  G  S     G  G     G  G
 11  S     G  G     G  G     G  G
 12  G  G     G  G     G  G  S
 13     G  G     G  G  S     G  G
 14  G     G  G  S     G  G     G
 15  G  G  S     G  G     G  G
 16  S     G  G     G  G     G  G
 17  G  G     G  G     G  G  S
 18     G  G     G  G  S     G  G
 19  G     G  G  S     G  G     G
 20  G  G  S     G  G     G  G
```

ここでは 100 ステップ施行した内の 20 ステップの実行結果を提示する。左はセルの数 10、車の数 4 での実行結果、右はセルの数 10、車の数 7 での実行結果である。左の実行結果を見ると、セルに対して車の数が少ない場合、車の状態は、十分時間が経過すると S がなくなり、G だけになることがわかる。また、右の実行結果を見ると Rule184 と同様、赤枠で囲った渋滞クラスターが進行方向とは逆に動いてゆくように見えるのを確認することができた。しかし、Quick-Start では、Rule184 とは異なり２台先の車の動きを見ているため、車２台が渋滞クラスターから出ると同時に後ろから２台入るという現象が繰り返されている。したがって、Quick-Start ではこの現象が繰り返され、渋滞クラスター全体が２つ後ろに移動したように見えることをシミュレーションにより確認することができた。

6 Slow-start

6-1 Slow-Start のルール

３つ目のルールは、「Slow-Start」というルールである。このルールでは、G と S に加えて W(wait)の状態も加えてシミュレーションしてゆく。G、S、W が表す意味は図５の状態遷移図の矢印の通りである。このルールは、動いている車は次の時刻も動きやすく、止まってしまった車は発進が鈍いという状態を表したルールである。ルール説明時には G、S、W の状態をそれぞれ「走っている」状態の車、「止まっている」状態の車、「待っている」状態の車と言い表すことにする。この３状態に分けてルールを設定する。このルールの条件は以下の通りである。

①周期境界条件

②車が「走っている」状態の時

②-1　１つ前のセルに車が存在しない場合、次のステップで進み、車の状態は変化しない。

②-2　１つ前のセルに車が存在する場合、次のステップで進むことはできず、車は「止まっている」状態に変化する。

③車が「止まっている」状態の時

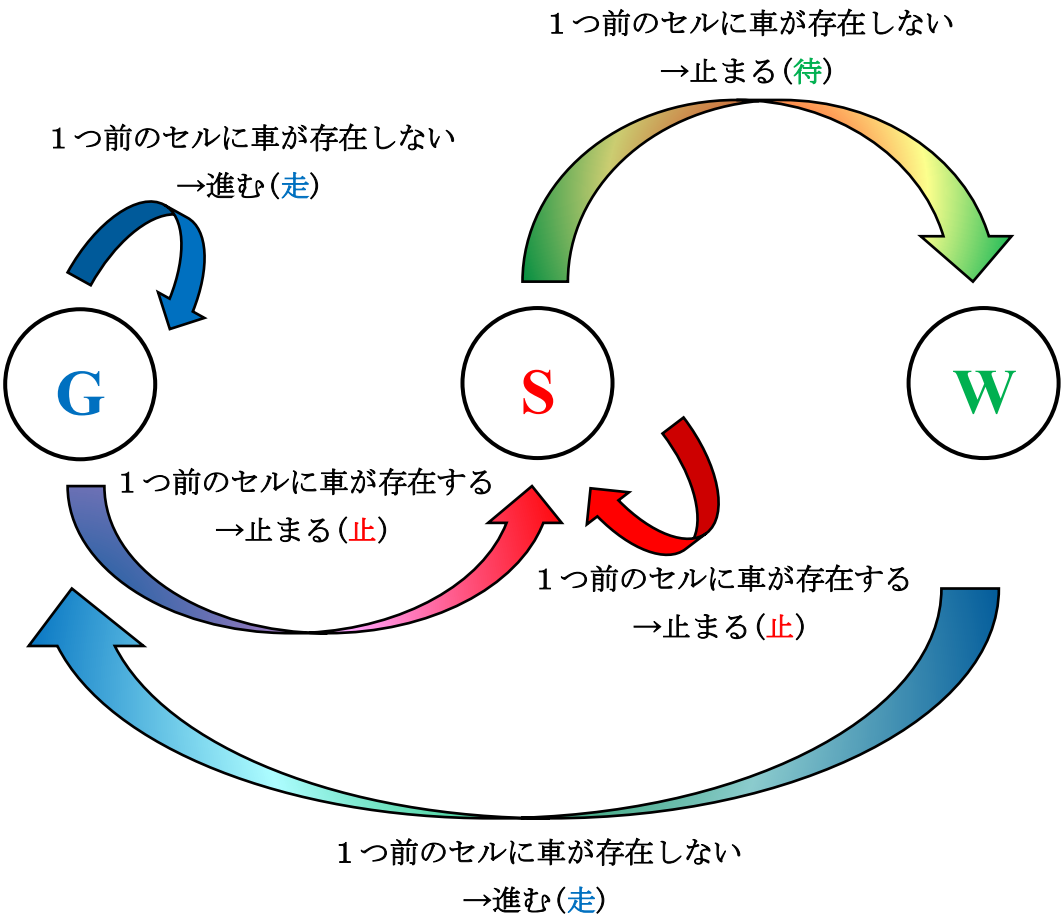
③-1　１つ前のセルに車が存在しない場合、次のステップで進むことはできず、車は「待っている」状態に変化する。

③-2　１つ前のセルに車が存在する場合、次のステップで進むことはできず、車の状態は変化しない。

④車が「待っている」状態の時

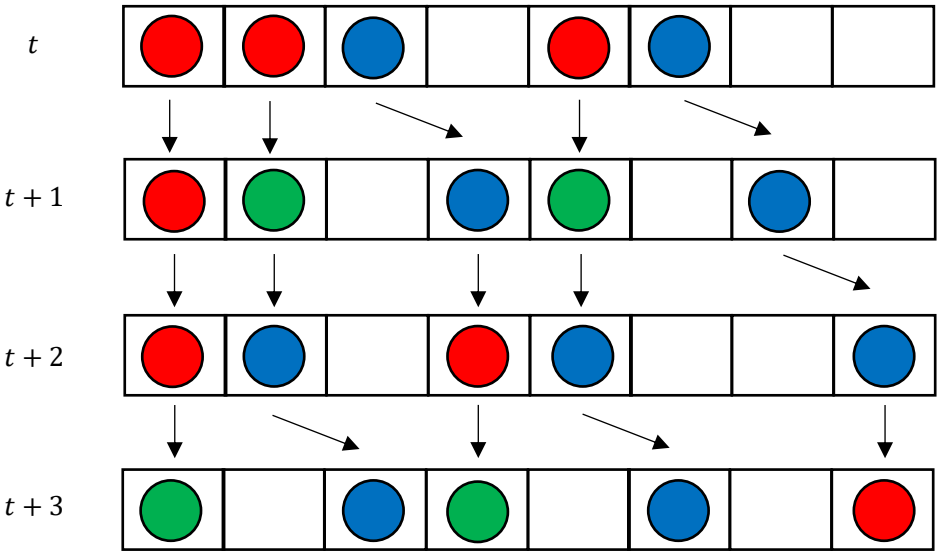
次のステップで車は「走っている」状態に変化する。

図５は Slow-Start の状態遷移図である。



【図５】Slow-Start　状態遷移図

図６は Slow-Start の動き方である。



●：「走っている」状態 / ●：「止まっている」状態 / ●：「待っている」状態

【図６】Slow-Start

このルールの下、C 言語でシミュレーションする。

6-2 Slow-Start のシミュレーション

以下は Slow-Start の C 言語によるシミュレーション･プログラム Slow-Start.c である。

プログラム Slow-Start.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(int argc, char **argv)
{
    int C;    //セルの数
    double M; //車の数

    if (argc != 3) {
        printf("セルの数：");
        scanf("%d",&C);
        printf("車の数：");
        scanf("%lf",&M);
    }
    else {
        C = atoi(argv[1]);
        M = atof(argv[2]);
    }
    //C個のセルを持つ配列を用意
```

```
    int n = 1;
    int a[C];
    int a2[C];
    char b[4]={' ','G','S','W'};
    // :車が存在しない
    //G:走っている状態
    //S:止まっている状態
    //W:待っている状態


    //初期化
    for(int i=0; i<C; i++){
        a[i] = 0;
        a2[i] = 0;
    }

    //M台の車をランダムに配置
    int value;
    value=0;
    srand(0);
    while(value<M){
        int r=rand()%C;   //ランダムに車を配置

        if(a[r]==0){      //rのセルが空いていれば
            a[r]=1;       //そのrのセルに1台配置する
            value=value+1;//M台になるまで続ける
    }
}

printf("%3d",n-1);
printf("");

for(int i=0; i<C; i++){
        printf("%3c",b[a[i]]);
        }
        printf("¥n");

    for(int x=0; x<100; x++){
        //Slow-Startの条件
        for(int i=C-1; i>=0; i--){
                if(i == C-1){
                    //周期境界条件(先頭)
                    if(a[C-1] == 1){
                        //車が存在するとき
                        if(a[0] == 0){
                            //前に車が存在しないとき
                            a2[C-1] = 0;
                            a2[0] = 1;
                        }else if(a[0] == 1 && a[1] == 0){
                            //前に車が存在しているが、2個前に車が存在しないとき
                            a2[C-1] = 3;
                        }else{
                            //前に車が存在しているかつ、2個前にも車が存在しているとき
                            a2[C-1] = 2;
```

```
        }
    }else if(a[C-1] == 2){
        //止まっている状態のとき
        if(a[0] == 0){
            //前に車が存在していないとき
            a2[C-1] = 1;
        }else if(a[0] == 1 && a[1] == 0){
            //前に車が存在しているが、2個前に車が存在していないとき
            a2[C-1] = 3;
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[C-1] = 2;
        }
    }
}else if(i == C-2){
    //周期境界条件(前から2つ目の場所)
    if(a[C-2] == 1){
        //車が存在するとき
        if(a[C-1] == 0){
            //前に車が存在していないとき
            a2[C-2] = 0;
            a2[C-1] = 1;
        }else if(a[C-1] == 1 && a[0] == 0){
            //前に車が存在しているが、2個前に車が存在していないとき
            a2[C-2] = 3;
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[C-2] = 2;
        }
    }else if(a[C-2] == 2){
        //止まっている状態のとき
        if(a[C-1] == 0){
            //前に車が存在していないとき
            a2[C-2] = 1;
        }else if(a[C-1] == 1 && a[0] == 0){
            //前に車が存在しているが、2個前に車が存在していないとき
            a2[C-2] = 3;
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[C-2] = 2;
        }
    }
}else if(i == 0){
    //最後尾(Slow-Startだと重複しないからなくても良い)
    if(a[0] == 1){
        //車が存在するとき
        if(a[1] == 0){
            //前に車が存在していないとき
            if(a[C-1] == 1){
                //先頭の車との重複防止
                a2[1] = 1;
            }else{
                a2[0] = 0;
                a2[1] = 1;
```

```
            }
        }else if(a[1] == 1 && a[2] == 0){
            //前に車が存在しているが、2個前に車が存在していないとき
            a2[0] = 3;
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[0] = 2;
        }
    }else if(a[0] == 2){
        //止まっている状態のとき
        if(a[1] == 0){
            //前に車が存在していないとき
            a2[0] = 1;
        }else if(a[1] == 1 && a[2] == 0){
            //前に車が存在しているが、2個前に車が存在していないとき
            a2[0] = 3;
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[0] = 2;
        }
    }
}else{
    //その他の場所
    if(a[i] == 1){
        //車が存在するとき
        if(a[i+1] == 0){
            //前に車が存在していないとき
            a2[i] = 0;
            a2[i+1] = 1;
        }else if(a[i+1] == 1 && a[i+2] == 0){
            //前に車が存在しているが、2個前に車が存在していないとき
            a2[i] = 3;
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[i] = 2;
        }
    }else if(a[i] == 2){
        //止まっている状態のとき
        if(a[i+1] == 0){
            //前に車が存在していないとき
            a2[i] = 1;
        }else if(a[i+1] == 1 && a[i+2] == 0){
            //前に車が存在しているが、2個前に車が存在していないとき
            a2[i] = 3;
        }else{
            //前に車が存在しているかつ、2個前にも車が存在しているとき
            a2[i] = 2;
        }
    }
}

if(a[i] == 3){
    //待っている状態から走っている状態に変化
    a2[i] = 1;
```

```
                }
            }

        for(int i=0; i<C; i++){
            a[i] = a2[i];
            a2[i] = 0;              //次の状態のセルを現在の状態に変える
        }
        printf("%3d" ,n++);         //ステップ数
        printf("");

        for(int i=0; i<C; i++){
            printf("%3c",b[a[i]]);//車が存在しない時は空欄、車が存在する時はG,S,Wの車の状態で表す
        }

        printf("¥n");
    }
}
```

以下はプログラム Slow-Start.c の実行結果である。

Slow-Start.c の実行結果

```
セルの数：10
車の数：4
  0  G  G  G        G
  1  S  W     G        G
  2  S  G        G        G
  3  W     G        G        G
  4  G        G        G        G
  5     G        G        G     W
  6        G        G     G  G
  7  G        G        G     W
  8     G        G        G  G
  9        G        G     W     G
 10  G        G        G  G
 11     G        G     W     G
 12        G        G  G        G
 13  G        G     W     G
 14     G        G  G        G
 15        G     W     G        G
 16  G        G  G        G
 17     G     W     G        G
 18        G  G        G        G
 19  G     W     G        G
 20     G  G        G        G
```

```
セルの数：10
車の数：7
  0  G  G  G     G  G     G  G
  1  S  W     G  W     G  W     G
  2  S  G     S  G     S  G     S
  3  W     G  W     G  W     G  S
  4  G     S  G     S  G     S  S
  5     G  W     G  W     G  S  W
  6     S  G     S  G     S  S  G
  7  G  W     G  W     G  S  W
  8  S  G     S  G     S  S  G
  9  W     G  W     G  S  W     G
 10  G     S  G     S  S  G     S
 11     G  W     G  S  W     G  W
 12     S  G     S  S  G     S  G
 13  G  W     G  S  W     G  W
 14  S  G     S  S  G     S  G
 15  W     G  S  W     G  W     G
 16  G     S  S  G     S  G     S
 17     G  S  W     G  W     G  W
 18     S  S  G     S  G     S  G
 19  G  S  W     G  W     G  W
 20  S  S  G     S  G     S  G
```

ここでは 100 ステップ施行した内の 20 ステップの実行結果を提示する。左はセルの数 10、車の数 4 での実行結果、右はセルの数 10、車の数 7 での実行結果である。左の実行結果を見ると、セルに対して車の数が少ない場合、車の状態は、十分時間が経過すると

S がなくなり、G だけになることがわかる。また、右の実行結果を見ると今まで同様、赤枠で囲った渋滞クラスターが進行方向とは逆に動いてゆくように見えるのを確認することができた。Slow-start では、上記２つのルールにはなかった「待っている」時間が発生する。１台車が渋滞クラスターから抜けると、次のステップで渋滞クラスターの先頭車は「待っている」状態になり、後ろから１台車が渋滞クラスターに入ってくる。つまり、渋滞クラスターの先頭車が「待っている」状態であるため、その次の１ステップまでその渋滞クラスターの位置は変わらずという現象を繰り返す。したがって、２ステップごとに渋滞クラスター全体が後ろに１つ移動したように見えたことをシミュレーションにより確認することができた。

7 Two-lane quick start

7-1 Two-lane quick start のルール

４つ目のルールは「Two-lane quick start」というルールである。今度は２車線道路で車線変更ができる条件をルールに入れて考える。このルールは左車線と右車線に分けてルールを設定する。このルールの条件は以下の通りである。

①周期境界条件
②左車線の場合
- ②-1　１つ前のセルに車が存在しない場合、次のステップで進む。
- ②-2　１つ前のセルに車が存在するが、２つ前のセルに車が存在しない場合、１つ前のセルが進むと見越して次のステップで進む。
- ②-3　１つ前と２つ前のセルに車が存在し、１つ前の車がウインカーを出している時(３つ前のセルに車が存在している状態で２つ前と１つ前のセルにも車が存在しているかつ、１つ前の車から見て右車線の斜め前と横に車が存在していない場合)、次のステップで進む。
- ②-4　１つ前と２つ前のセルに車が存在しているかつ、右車線の斜め前と横に車が存在していないならば、次のステップで車線変更して右車線の斜め前へ進む。
- ②-5　１つ前と２つ前のセルに車が存在しているかつ、右車線の斜め前と横に車が存在しているならば、次のステップで進むことはできない。

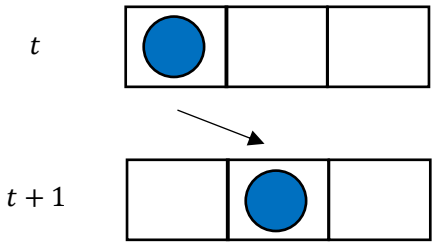
③右車線の場合
- ③-1　１つ前のセルに車がいないかつ左車線の斜め前と横に車がいない時、次のステップで左車線の斜め前に進む。
- ③-2　１つ前のセルに車が存在しない場合、次のステップで進む。
- ③-3　１つ前のセルに車が存在するが、２つ前のセルに車が存在しない場合、１つ前のセルが進むと見越して次のステップで進む。
- ③-4　１つ前と２つ前のセルに車が存在し、１つ前の車がウインカーを出している時

(３つ前のセルに車が存在している状態で２つ前と１つ前のセルにも車が存在しているかつ、１つ前の車から見て左車線の斜め前と横に車が存在していない場合)、次のステップで進む。

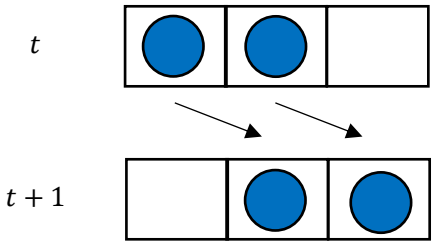
③-5　１つ前と２つ前のセルに車が存在しているかつ、左車線の斜め前と横に車が存在していないならば、次のステップで車線変更して左車線の斜め前へ進む。

③-6　１つ前と２つ前のセルに車が存在しているかつ、左車線の斜め前と横に車が存在しているならば、次のステップで進むことはできない。

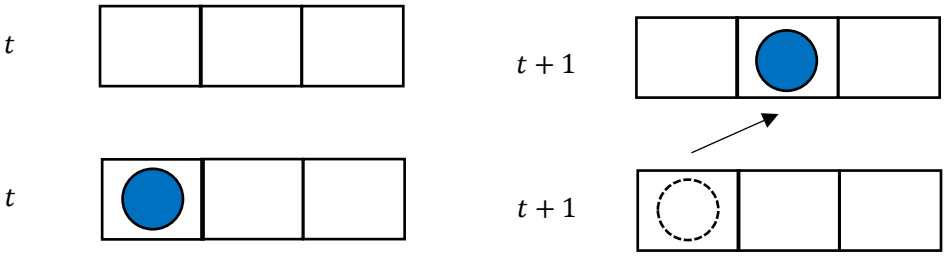
図７、図８、図９、図 10、図 11 は Two-lane quick start の動き方である。



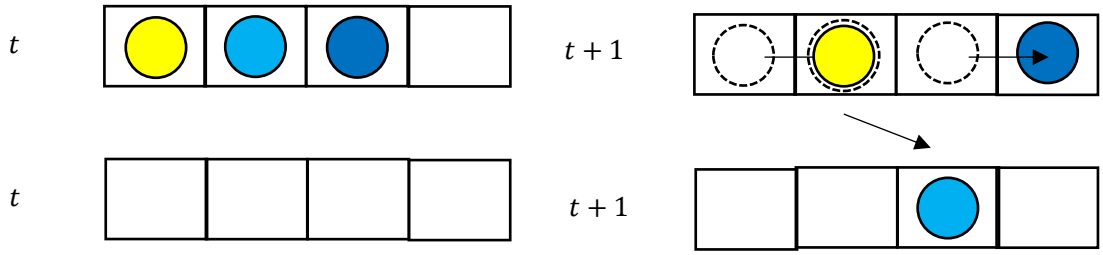
【図７】②-1 / ③-2 の動き方



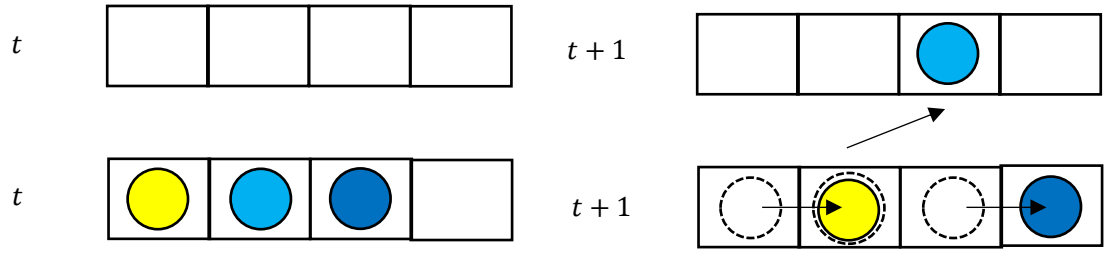
【図８】②-2 / ③-3 の動き方



【図９】③-1 の動き方



【図 10】 ：②-3 / ：②-4 の動き方



【図 11】 ：③-4 / ：③-5 の動き方

このルールの下、C 言語でシミュレーションする。

7-2 Two-lane quick start のシミュレーション

以下は Two-lane quick start の C 言語によるシミュレーション・プログラム Two-lane quick start.c である。

プログラム Two-lane quick start.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(int argc, char **argv){
    int C;   //セルの数
    double M;//車の数

    if (argc != 3) {
        printf("セルの数：");
        scanf("%d",&C);
        printf("車の数：");
        scanf("%lf",&M);
    }
    else {
        C = atoi(argv[1]);
        M = atof(argv[2]);
    }
    //C個のセルを持つ配列を用意
    int n = 1;
    int a[C];
    int a2[C];
    int z[C];
    int z2[C];
    char b[4]={'O','G','S','W'};
    // :車が存在しない
    //G:走っている状態
    //S:止まっている状態
    //W:待っている状態
```

```
//初期化
for(int i=0; i<C; i++){
    a[i]=0;
    a2[i]=0;
    z[i]=0;
    z2[i]=0;
}

//M台の車をランダムに配置
int value;
value=0;
srand(0);
while(value<M){
    while(value<(M/2)){
        int r=rand()%C;   //ランダムに車を配置
        if(a[r]==0){      //rのマスが空いていれば
            a[r]=1;       //そのrのマスに1台配置する
            value=value+1;//M台になるまで続ける
        }
    }
    int r=rand()%C;       //ランダムに車を配置
    if(z[r]==0){          //rのマスが空いていれば
        z[r]=1;           //そのrのマスに1台配置する
        value=value+1;    //M台になるまで続ける
    }
}

printf("%3d",n-1);
printf("");

for(int i=0; i<C; i++){
    printf("%2c",b[a[i]]);
}
printf("¥n   ");
for(int i=0; i<C; i++){
    printf("%2c",b[z[i]]);
}
printf("¥n¥n");

for(int x=0; x<100; x++){
    //Two-lane quick startの条件
    for(int i=C-1; i>=0; i--){
        if(i==C-1){
            //周期境界条件(先頭)
            if(a[C-1] == 1 ||a[C-1] == 2){
                //車が存在するとき
                if(a[0] == 0 || (a[0] == 1 && a[1] == 0) || (a[0] == 2 && a[1] == 0)){
                    //前に車が存在していない時or前に車が存在するが2個前に車が存在していない時前に進む
                    a2[0] = 1;
                }else if(a[2] != 0 && (a[1] == 1 || a[1] == 2) && z[1] == 0 && z[0] == 0){
                    //前の車が車線変更することを見越して前に進む
                    a2[0] = 1;
                }else if((a[0] == 1 || a[0] == 2)&& z[0] == 0 && z[C-1] == 0){
```

```
            //前に車が存在しているが、斜め前と隣に車が存在していない時、車線変更
            z2[0]=1;
        }else{
            //前にも斜め前にも進めない時、停車
            a2[C-1] = 2;
        }
    }
    if(z[C-1] == 1 ||z[C-1] == 2){
        //車が存在するとき
        if(z[0] == 0 && a[0] == 0 && a[C-1] == 0){
            //周りに車が前に存在していなければ、左車線に寄る
            a2[0] = 1;
        }else if(z[0] == 0 || (z[0] == 1 && z[1] == 0) || (z[0] == 2 && z[1] == 0)){
            //前に車が存在していない時or前に車が存在するが2個前に車が存在していない時前に進む
            z2[0] = 1;
        }else if(z[2] != 0 && (z[1] == 1 || z[1] == 2) && a[1] == 0 && a[0] == 0){
            //前の車が車線変更することを見越して前に進む
            z2[0] = 1;
        }else if((z[0] == 1 || z[0] == 2) && a[0] == 0 && a[C-1] == 0){
            //前に車が存在しているが、斜め前と隣に車が存在していない時、車線変更
            a2[0] = 1;
        }else{
            //前にも斜め前にも進めない時、停車
            z2[C-1] = 2;
        }
    }
}else if(i==C-2){
    //周期境界条件（前から2つ目の場所）
    if(a[C-2] == 1 || a[C-2] == 2){
        //車が存在するとき
        if(a[C-1] == 0 || (a[C-1] == 1 && a[0] == 0) || (a[C-1] == 2 && a[0] == 0)){
            //前に車が存在していない時or前に車が存在するが2個前に車が存在していない時前に進む
            a2[C-1] = 1;
        }else if(a[1] != 0 && (a[0] == 1 || a[0] == 2) && z[0] == 0 && z[C-1] == 0){
            //前の車が車線変更することを見越して前に進む
            a2[C-1] = 1;
        }else if((a[C-1] == 1 || a[C-1] == 2) && z[C-1] == 0 && z[C-2] == 0){
            //前に車が存在しているが、斜め前と隣に車が存在していない時、車線変更
            z2[C-1] = 1;
        }else{
            //前にも斜め前にも進めない時、停車
            a2[C-2] = 2;
        }
    }
    if(z[C-2] == 1 || z[C-2] == 2){
        //車が存在するとき
        if(z[C-1] == 0 && a[C-1] == 0 && a[C-2] == 0){
            //周りに車が存在していなければ、左車線に寄る
            a2[C-1]=1;
        }else if(z[C-1] == 0 || (z[C-1] == 1 && z[0] == 0) || (z[C-1] == 2 && z[0] == 0)){
            //前に車が存在していない時or前に車が存在するが2個前に車が存在していない時前に進む
            z2[C-1] = 1;
        }else if(z[1] != 0 && (z[0] == 1 || z[0] == 2) && a[0] == 0 && a[C-1] == 0){
            //前の車が車線変更することを見越して前に進む
```

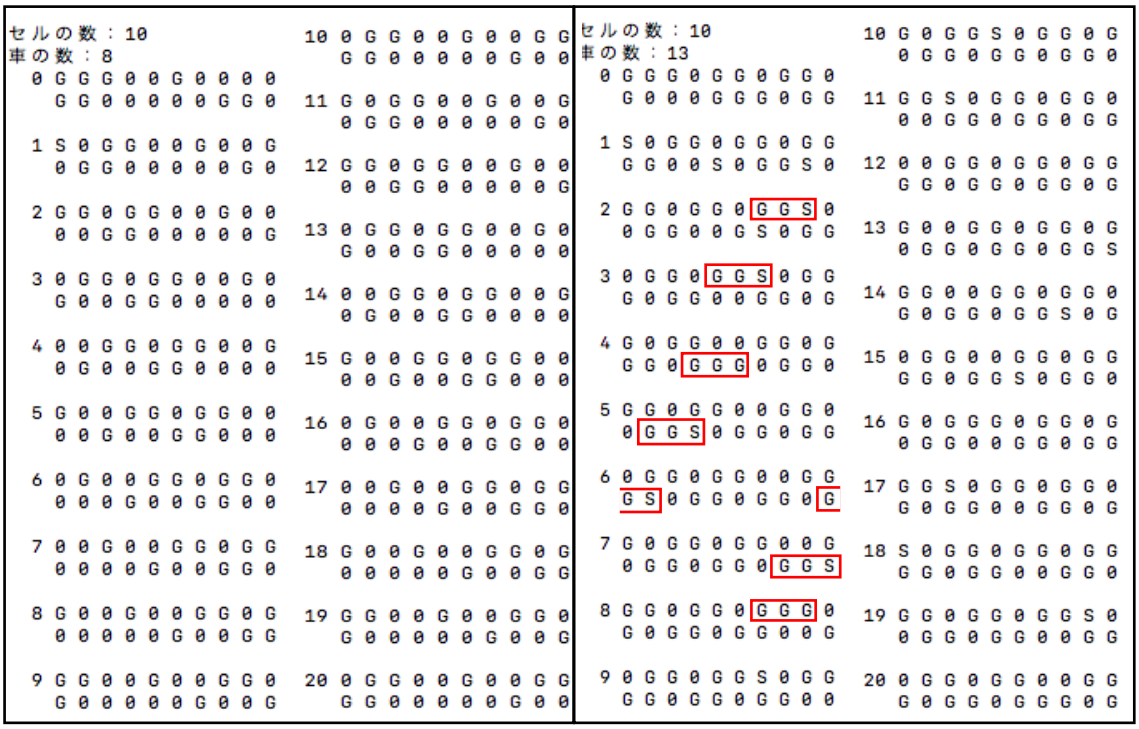
```
            z2[C-1] = 1;
        }else if((z[C-1] == 1 || z[C-1] == 2 ) && a[C-1] == 0 && a[C-2] == 0){
            //前に車が存在しているが、斜め前と隣に車が存在していない時、車線変更
            a2[C-1] = 1;
        }else{
            //前にも斜め前にも進めない時、停車
            z2[C-2] = 2;
        }
    }
}else if(i==C-3){
    //周期境界条件（前から2つ目の場所）
    if(a[C-3] == 1 || a[C-3] == 2){
        //車が存在するとき
        if(a[C-2] == 0 || (a[C-2] == 1 && a[C-1] == 0) || (a[C-2] == 2 && a[C-1] == 0)){
            //前に車が存在していない時or前に車が存在するが2個前に車が存在していない時前に進む
            a2[C-2] = 1;
        }else if(a[0] != 0 && (a[C-1] == 1 || a[C-1] == 2) && z[C-1] == 0 && z[C-2] == 0){
            //前の車が車線変更することを見越して前に進む
            a2[C-2] = 1;
        }else if((a[C-2] == 1 || a[C-2] == 2) && z[C-2] == 0 && z[C-3] == 0){
            //前に車がいるが、斜め前と隣に車が存在していない時、車線変更
            z2[C-2] = 1;
        }else{
            //前にも斜め前にも進めない時、停車
            a2[C-3] = 2;
        }
    }
    if(z[C-3] == 1 || z[C-3] == 2){
        //車が存在するとき
        if(z[C-2] == 0 && a[C-2] == 0 && a[C-3] == 0){
            //周りに車が存在していなければ、左車線に寄る
            a2[C-2]=1;
        }else if(z[C-2] == 0 || (z[C-2] == 1 && z[C-1] == 0) || (z[C-2] == 2 && z[C-1] == 0)){
            //前に車が存在していない時or前に車が存在するが2個前に車が存在していない時前に進む
            z2[C-2] = 1;
        }else if(z[0] != 0 && (z[C-1] == 1 || z[C-1] == 2) && a[C-1] == 0 && a[C-2] == 0){
            //前の車が車線変更することを見越して前に進む
            z2[C-2] = 1;
        }else if((z[C-2] == 1 || z[C-2] == 2 ) && a[C-2] == 0 && a[C-3] == 0){
            //前に車がいるが、斜め前と隣に車が存在していない時、車線変更
            a2[C-2] = 1;
        }else{
            //前にも斜め前にも進めない時、停車
            z2[C-3] = 2;
        }
    }
}else if(i == 0){
    //最後尾
    if(a[0] == 1 || a[0] == 2){
        //車が存在するとき
        if(a[1] == 0 || (a[1] == 1 && a[2] == 0) || (a[1] == 2 && a[2] == 0)){
            //前に車が存在していない時or前に車が存在するが2個前に車が存在していない時前に進む
            a2[1] = 1;
        }else if(a[3] != 0 && (a[2]==1||a[2]==2) && z[2]==0 && z[1]==0){
```

```
                    a2[1]=1;
                }else if((a[1]==1||a[1]==2) && z[1]==0 && z[0]==0){
                    //前に車が存在しているが、斜め前と隣に車が存在していない時、車線変更
                    z2[1]=1;
                }else{
                    //前にも斜め前にも進めない時、停車
                    a2[0] = 2;
                }
            }
            if(z[0] == 1 || z[0] == 2){
                //車が存在するとき
                if(z[1]==0 && a[1]==0 && a[0]==0){
                    //周りに車が存在していなければ、左車線に寄る
                    a2[1]=1;
                }else if(z[1] == 0 || (z[1] == 1 && z[2] == 0) || (z[1] == 2 && z[2] == 0)){
                    //前に車が存在していないとき or 前に車が存在しているが、2個前に車が存在していない
時、前に進む
                    z2[1] = 1;
                }else if(z[3] != 0 && (z[2] == 1 || z[2] == 2) && a[2] == 0 && a[1] == 0){
                    //前の車が車線変更することを見越して前に進む
                    z2[1] = 1;
                }else if((z[1] == 1 || z[1] == 2) && a[1] == 0 && a[0] == 0){
                    //前に車がいるが、斜め前と隣に車が存在していない時、車線変更
                    a2[1] = 1;
                }else{
                    //前にも斜め前にも進めない時、停車
                    z2[0] = 2;
                }
            }
        }else{
            //その他の場所
            if(a[i] == 1 || a[i] == 2){
                //車が存在するとき
                if(a[i+1] == 0 || (a[i+1] == 1 && a[i+2] == 0) || (a[i+1] == 2 && a[i+2] == 0)){
                    //前に車が存在していない時or前に車が存在するが2個前に車が存在していない時前に進む
                    a2[i+1] = 1;
                }else if(a[i+3] != 0 && (a[i+2] == 1 || a[i+2] == 2) && z[i+2] == 0 && z[i+1] == 0){
                    //前の車が車線変更することを見越して前に進む
                    a2[i+1] = 1;
                }else if((a[i+1] == 1 || a[i+1] == 2) && z[i+1] == 0 && z[i] == 0){
                    //前に車が存在しているが、斜め前と隣に車が存在していない時、車線変更
                    z2[i+1] = 1;
                }else{
                    //前にも斜め前にも進めない時、停車
                    a2[i] = 2;
                }
            }
            if(z[i] == 1 || z[i] == 2){
                //車が存在するとき
                if(z[i+1] == 0 && a[i+1] == 0 && a[i] == 0){
                    //周りに車が存在していなければ、左車線に寄る
                    a2[i+1] = 1;
                }else if(z[i+1] == 0 || (z[i+1] == 1 && z[i+2] == 0) || (z[i+1] == 2 && z[i+2] == 0)){
                //前に車が存在していない時or前に車が存在しているが2個前に車が存在していない時前に進む
```

```
                    z2[i+1] = 1;
                }else if(z[i+3] != 0 && (z[i+2] == 1 || z[i+2] == 2) && a[i+2] == 0 && a[i+1] == 0){
                    //前の車が車線変更することを見越して前に進む
                    z2[i+1] = 1;
                }else if((z[i+1] == 1 || z[i+1] == 2) && a[i+1] == 0 && a[i] == 0){
                    //前に車がいるが、斜め前と隣に車が存在していない時、車線変更
                    a2[i+1] = 1;
                }else{
                    //前にも斜め前にも進めない時、停車
                    z2[i] = 2;
                }
            }
        }
    }

    for(int i=0; i<C; i++){
        a[i] = a2[i];//次の状態のセルを現在の状態に変える
        z[i] = z2[i];
        a2[i] = 0;
        z2[i] = 0;
    }

    printf("%3d" , n++);//ステップ数
    printf("");

    for(int i=0; i<C; i++){
        printf("%2c",b[a[i]]);
    }
    printf("¥n   ");
    for(int i=0; i<C; i++){
        printf("%2c",b[z[i]]);
    }
    printf("¥n¥n");
  }
}
```

以下はプログラム Two-lane quick start.c の実行結果である。

Two-lane quick start.c の実行結果



ここでは 100 ステップ施行した内の 20 ステップの実行結果を提示する。左はセルの数 10（1 車線あたり 10 セル）、車の数 8 での実行結果、右はセルの数 10（1 車線あたり 10 セル）、車の数 13 での実行結果である。左の実行結果を見ると、セルに対して車の数が少ない場合、車の状態は、十分時間が経過すると S がなくなり、G だけになることがわかる。また、右の実行結果を見ると今まで同様、赤枠で囲った渋滞クラスターが進行方向とは逆に動いてゆくように見えるのを確認することができた。Two-lane quick start では、Quick-start と同じ 1 台先の動きを見通している条件を入れているため、基本は 2 台車が渋滞クラスターから抜けると後ろから 2 台車が渋滞クラスターに入ってくるという現象を繰り返し、クラスター全体が後ろに 2 つ移動したように見えた。しかし、この条件下では渋滞クラスターに入って止まってしまっても、周りの車の状況によって車線変更することができるため、先頭車でなくても渋滞クラスターから抜けられる場合がある。

8 まとめ

「1 はじめに」で述べた、渋滞を現実に起こっている「Rule184」、「Quick-Start」、「Slow-Start」、「Two-lane quick start」の 4 つの条件下で再現するという目標を達成することができた。この 4 つの条件下において、全ての条件下で渋滞クラスターが時間の経過ごと

に進行方向とは逆に動いてゆくように見えることを確認することができた。しかし、そのそれぞれのルールにおける渋滞クラスターの発生の仕方や動き方にそれぞれ違いが見られた。「Quick-Start」では、渋滞クラスター全体が２つ後ろに移動するため、渋滞から早く抜けることができ、「Two-lane quick start」では先頭車でなくても渋滞クラスターから抜けられる条件から早く渋滞から抜けられる可能性もあった。しかし、全ての車のドライバーが１つのルール通りに動くとは限らない。すなわち、今回の研究の結果から、この様々な条件を組み合わせてシミュレーションすることにより、もっと現実に近い渋滞現象を再現することができると考えた。したがって、今後の課題としては、複数のルールを組み合わせたシミュレーションを行い、より現実に近い渋滞の現象をシミュレーションし再現するとともに、より早く渋滞から解放される方法を探ることである。

9 謝辞

最後になりましたかが、今回の卒業研究にあたり、最後までサポートしてくださった明治大学 総合数理学部 現象数理学科 桂田祐史先生、本研究を行うにあたり参考にさせていただいた西成活裕先生、桂田研究室の先輩方には厚くお礼申し上げます。

10 参考文献

[1]西成活裕 (2006) 「渋滞学」 新潮社