Algorithmus

VMax: 5; P-DD: 33; Cruise control: Off; Increased delay at v=0: 0 (Off);

Runde					6. **T-jam**
1	1. INITIAL/**NewTick**	2. **Beschleunigen**	3. **Bremsen**	4. **Trödeln**	5. **Fahren**	
	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	1
	A----B---C---D-E----	A----B---C---D-E----	A----B---C---D-E----	A----B---C---D-E----	----A---B---CD--E---	
	5----4---2---1-1----	5----5---3---2-2----	4----3---3---1-2----	4----3---3---0-1----	----4---3---30--1---	
	--------------------	=----->--->--->-->----	<----<--->---=->----	<----<--->---<-=----	----<---<--->!--=---	
2	1. **NewTick**	2. **Beschleunigen**	3. **Bremsen**	4. **Trödeln**	5. **Fahren**	
	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	1
	----A---B---CD--E---	----A---B---CD--E---	----A---B---CD--E---	----A---B---CD--E---	-------A--B-CD---E--	
	----4---3---30--1---	----5---4---41--2---	----3---3---01--2---	----3---2---00--1---	-------3--2-00---1--	
	--------------------	---->--->--->>-->---	----<---=---<>-->---	----<---<---<=--=---	-------<--<-!!---=--	
3	1. **NewTick**	2. **Beschleunigen**	3. **Bremsen**	4. **Trödeln**	5. **Fahren**	
	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	2
	-------A--B-CD---E--	-------A--B-CD---E--	-------A--B-CD---E--	-------A--B-CD---E--	---------AB-C-D----E	
	-------3--2-00---1--	-------4--3-11---2--	-------2--1-01---2--	-------2--0-01---2--	---------20-0-1----2	
	--------------------	------->-->->>--->--	-------<--<-=>--->--	-------<--<-=>--->--	---------<!-!->---->	
4	1. **NewTick**	2. **Beschleunigen**	3. **Bremsen**	4. **Trödeln**	5. **Fahren**	
	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	2
	---------AB-C-D----E	---------AB-C-D----E	---------AB-C-D----E	---------AB-C-D----E	-E-------A-BC--D----	
	---------20-0-1----2	---------31-1-2----3	---------01-1-2----3	---------01-0-1----2	-2-------0-10--1----	
	--------------------	--------->>->->---->	---------<>->->---->	---------<>-=-=----=	-=-------!->!--=----	
5	1. **NewTick**	2. **Beschleunigen**	3. **Bremsen**	4. **Trödeln**	5. **Fahren**	
	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	2
	-E-------A-BC--D----	-E-------A-BC--D----	-E-------A-BC--D----	-E-------A-BC--D----	----E-----AB-C---D--	
	-2-------0-10--1----	-3-------1-21--2----	-3-------1-01--2----	-3-------1-01--2----	----3-----10-1---2--	
	--------------------	->------->->>-->----	->------->-<>-->----	->------->-<>-->----	---->----->!->--->--	

KONSTANTEN

```
OCAR aCars[990]
```

STATUS (Settings)

```
Cars:   {1..990}
VMax:   {1..50}
P-DD:   {0..90}
Mode:   {Step1|Step6|Auto6|AutoX}
Cruise-control:          {On|Off}
Increased delay at v=0: {0..90}
Save to file:            {On|Off}
File:                    -..[.bmp]-
```

STATUS (Gaugings)

```
Ticks:       {0..100%}  {0..500}
Total   traffic jams:   {0..}
Current traffic jams:   {0..}
Time    without jams:   YY,YYYsec
```

KEYEVENTS

MENU(default)

e – Exit
m – Toggle Mode
q – Start..
f – Save..
c – Toggle Cruise-control

MENU(start/settings)

e – Exit
s – Abort
+ – Inkrement Setting
- – Dekrement Setting
TAB – Next.. **TAB+Alt** – Prev..
ENTER – SaveSettings
0..9 – add cypher
BACKSPACE – remove cypher

MENU(animation)

e – Exit
s – Abort
LEER – Next.. [MODE==STEP1||MODE==STEP6]
p – Abort and print [SAVE==ON]

MENU(save/settings)

Esc – Abort
> – Toggle ON/OFF
ENTER – Save..
{a-z,A-Z,0-9,\,., ,_,-} – Append
BACKSPACE – Remove
F1|F2|F3 - Toggle Color Schemes

Beschreibungen

Straße:
Streifen a **1000 Autos lang**. Straße ist **endlich, aber unbegrenzt**, denn ihr rechtes und linkes Ende sind topologisch miteinander verbunden. Auf der **einspurigen Straße** fahren Autos nach rechts.

Cruise-control (Tempomat):
Auf **Trödeln** bei Maximalgeschwindigkeit [**V == Vmax**] **verzichten.**

Increased delay at v=0:
Wenn [**iIncreasedDelayAtV0Prozent > 0**], dann:
Wenn im Zustand '**NewTick**' [**v == 0**] benutze in dieser Runde **X als Trödelwahrscheinlichkeit** im Zustand '**Trödeln**'. => Der Fahrer trödelt dann beim anfahren stärker.
(X∈{0..99}:X>iPDDProzent:X=iPDDProzent+(100-X)*iIncreasedDelayAtV0Prozent)

Save to file/Save State:
Wenn [**SAVE==ON**] wird immer nach dem Zustand '**Teste Stau**' der **Zustand der Autos gespeichert.**
Wenn, **p – Abort and print [SAVE==ON],** gedrückt wird, wird der Algorithmus abgebrochen und alle bisher **gespeicherten Zustände** als **Farbschema** in eine ***.bmp** Datei gespeichert.
Falls der Algorithmus abgebrochen wird aufgrund von [**Ticks >= MAXTICKS**] und [**SAVE==ON**] werden auch die bisher **gespeicherten Zustände** als **Farbschema** in eine Datei (***.bmp**) gespeichert.
Beim Speichern in die Datei (***.bmp**) werden immer **die gespeicherten Zustände** in ein **Farbschema** umgewandelt.

Ticks:
Anzahl an Ticks/Runden. Maximal **500** (**MAXTICKS)** danach wird automatisch abgebrochen und falls [**SAVE==ON**] werden auch die **bisher gespeicherten Zustände** als **Farbschema** in eine Datei (***.bmp**) gespeichert. **Ein Tick** umfasst die Zustände '**NewTick**' > '**Beschleunigen**' > '**Bremsen**' > '**Trödeln**' > '**Fahren**' > '**Teste Stau**'

P-DD (Dilly-dally-P%):
Trödelwahrscheinlichkeit (**iPDDProzent bzw X**).
Wenn [**Random(0..100) < P-DD**] dann trödelt das Auto und es wird um 1 langsamer.

Total traffic jams + Current traffic jams + Time without jams:

Total traffic jams ist die Anzahl gesamter **neuer Staus.**
Current traffic jams ist die Anzahl **gerade bestehender Staus.**

FALL A:
Ein **neuer Stau** ist dadurch zu erkennen, dass ein Auto, welches
noch nicht im Stau war, nach dem Zustand '**Fahren**',
die **Geschwindigkeit 0** hat
UND entweder, der **Abstand zum Vordermann größer 1** ist
ODER der **Vordermann nicht im Stau** ist.

FALL B:
Ein Auto **wird teil eines bestehenden Staus**, wenn das Auto
noch nicht im Stau ist, nach dem Zustand '**Fahren**',
die **Geschwindigkeit 0** hat,
dessen **Abstand zum Vordermann kleiner 2** ist
UND der **Vordermann im Stau** ist.

FALL C:
Ein Auto **wird teil eines bestehenden Staus**, wenn das Auto
noch nicht im Stau ist, nach dem Zustand '**Fahren**',
die **Geschwindigkeit größer 0** hat,
dessen **Abstand zum Vordermann kleiner 2** ist,
UND der **Vordermann im Stau** ist.

FALL D:
Ein Auto ist **aus dem Stau draußen**, wenn das Auto
im Stau ist, nach dem Zustand '**Fahren**',
die **Geschwindigkeit größer 0** hat
UND dessen **Abstand zum Vordermann kleiner gleich der eigenen Geschwindigkeit** ist.

Im **Run Mode AUTO6|AUTOX** wird die Zeit '**Time without jams**' gemessen.
(wie lange es dauert bis der erste Stau auftritt/Wie lange noch gar kein Stau da ist).

Run mode STEP1:
Im **STEP1** Modus werden pro Runde/Tick nacheinander auf Tastendruck '**LEER**' (**Leertaste**) die Zustände '**NewTick**' > '**Beschleunigen**' > '**Bremsen**' > '**Trödeln**' > '**Fahren**' > '**Teste Stau**' ausgegeben.
Die Zeit ohne Staus wird nicht gemessen.

Run mode STEP6:
Im **STEP1** Modus werden pro Runde/Tick nacheinander die Zustände '**NewTick**' > '**Beschleunigen**' > '**Bremsen**' > '**Trödeln**' > '**Fahren**' > '**Teste Stau**' durchloffen aber nur der Zustand '**Teste Stau**' ausgegeben.
Dann wird auf ein Tastendruck '**LEER**' (**Leertaste)** gewartet.
Die Zeit ohne Staus wird nicht gemessen.

Run mode AUTO6:
Im **AUTO6** Modus wird wie bei **STEP6** nur noch pro Runde/Tick der Zustand '**Teste Stau**' ausgegeben aber nicht auf einen Tastendruck gewartet.
Die Zeit wird gemessen, wie lange es dauert bis der erste Stau auftritt.

Run mode AUTOX:
Im **AUTOX** Modus wird pro Runde/Tick nur noch die Messung (Gaugings) aktualisiert.
Es wird nicht auf einen Tastendruck gewartet.
Die Zeit wird gemessen, wie lange es dauert bis der erste Stau auftritt.

Run mode STEP1+STEP6+AUTO6+AUTOX:
Falls [**Ticks >= MAXTICKS**] wird der Algorithmus abgebrochen.
Falls der Algorithmus abgebrochen wird aufgrund von [**Ticks >= MAXTICKS**] und [**SAVE==ON**] werden auch die **bisher gespeicherten Zustände** in eine Datei (***.bmp**) als **Farbschema** gespeichert.
In allen Modi kann mit '**s**' abgebrochen oder mit '**e**' das Programm beendet werden.
Falls [**SAVE=ON**] wird mit '**p**' abgebrochen und die **bisher gespeicherten Zustände** als ***.bmp** in ein **Farbschema** umgerechnet und in eine Datei gespeichert.

MockUps

```
45x5
*Cars:***.XX****Increased delay at v=0:**.X%*
*Vmax:****.X****Cruise-control:**********OFF*
*P-DD:***.X%****Save to file:***********OFF*
*Mode:*Step1****-test01234567890123..[.bmp]-*
*Color Schemes:*{F1,F2,F3,F4,F5,F6,F7,F8,F9}*

34x5
*Ticks:**********.XX%******.XX****
*Total   traffic jams:*.XXXXXX****
*Current traffic jams:*.XXXXXX****
*Time    without jams:***0,000sec*
**********************************


        32x11
        ********************************
        **Cars:**.XX***[+/-]**{1..990}**
        ********************************
        **Vmax:***.X**********************
        ********************************
        **P-DD:**.X%**********************
        ********************************
        **IDV0:**.X%**********************
        ********************************
        **Save Settings? [ENTER]********
        ********************************
```

Datenstrukturen

```
struct object_car {
   int iV;
   int iVChange;
   int iJamGroupId;
   int iPosition;
   bool bIsInJam;
   bool bDoDelayAtV0;
};

enum states {
   newtick = 0,
   accelerate,
   retard,
   dilly_dally,
   drive,
   test_jam
};

struct settings {
   int iVMax;
   int iCars;
   int iPDDProzent;
   enum toggle eTCruiseControl;
   int iIncreasedDelayAtV0Prozent;
   enum toggle eTSaveToFile;
   enum mode eMode;
   char acFilename[COMPLPATHLENGTH];
   char acComplFilePath[COMPLPATHLENGTH];
   struct colorschemes* asCSchemes[9];
};
```

```
enum toggle { on, off };

struct gaugings {
   int iTicks;
   int iTotalTrafficJams;
   int iCurrentTrafficJams;
   clock_t runtime;
   char acTimeStamp[TIMESTAMPLENGTH];
   struct saveState ppsState[990][MAXTICKS];
};

struct naschmodell {
   struct settings sSettings;
   struct gaugings sGaugings;
   struct object_car asCars[990];
};

enum mode {
   step1 = 0, step6, auto6, autoX
};

struct saveState {
   bool bIsInJam;
   int iVTotal;
   int iJamGroup;
   int iPosition;
};

struct colorschemes {
 unsigned int iId;
 COLOR(*pf_Converter)(struct saveState*, struct settings*);
};
```

Algorithmus

```
do newtick:
    iV = iV + iVChange;
    iVChange = 0;
    [settings.iIncreasedDelayAtV0Prozent > 0 && iV + iVChange== 0] { bDoDelayAtV0 = true; }

print newtick
userinteractionSTEP1

do accelerate:
    [ iV + iVChange < settings.iVMax ] { iVChange++; }

print accelerate
userinteractionSTEP1

do retard:
    [iV + iVChange > diff(myCar.iPosition, carInFront.iPosition)]
    { iVChange = diff(my.iPosition, carInFront.iPosition) – iV; }

print retard
userinteractionSTEP1
```

```
do dilly_dally:
    [ iV + iVChange > 0 ]
    {
       [settings.iIncreasedDelayAtV0Prozent > 0 && bDoDelayAtV0 == true]
       {
          bDoDelayAtV0 = false;
          [Random(0..100) <
            (settings.iPDDProzent + ((100 – settings.iIncreasedDelayAtV0Prozent) * 10 / 100))]
          {
             iVChange--;
          }
       }
       else [settings.eTCruiseControl == on && iV + iVChange >= settings.IVMax]
       {
          //Tempomat is on => no dilly_dally
       }
       else [Random(0..100) < settings.iPDDProzent]
       {
          iVChange--;
       }
    }

print dilly_dally
userinteractionSTEP1

do drive:
   iPosition = (iPosition + iV + IVChange) % STREET_LENGTH;

print drive
userinteractionSTEP1
```

do test_jam

```
   gaugings.iTicks++;

   for ( int i = 0 ; i < settings.iCars ; i++ )
      do_test_jam_OnOne(i);

   int iLastJamGroupId = asCars[settings.iCars-1].iJamGroupId;
   bool bIsAtLeastOneInJam = false;
   gaugings.iCurrentTrafficJams = 0;
   for ( int i = 0 ; i < settings.iCars ; i++ )
   {
      [asCars[i].bIsInJam == true]
      {
         bIsAtLeastOneInJam = true;
         [asCars[i].iJamGroupId != iLastJamGroupId]
         {
            gaugings.iCurrentTrafficJams++;
            iLastJamGroupId = asCars[i].iJamGroupId;
         }
      }
   }
   [ bIsAtLeastOneInJam == true &&  gaugings.iCurrentTrafficJams == 0 ] {
       //SELTENER FALL: all are in same jam
       gaugings.iCurrentTrafficJams++;
   }

   [settings.eTSaveToFile == on]
   {
      for ( int i = 0 ; i < settings.iCars ; i++ )
      {
         saveCarState(gaugings.iTicks, gaugings.ppsState, asCars, i);
      }
   }
```

print test_jam + print gaugings
userinteraction

do_test_jam_OnOne(int iCarId):

```
   [bIsInJam == true] {
      [iV + IVChange > 0 && iV + IVChange <= iDiff] {
         //FALL D
         bIsInJam = false;
         iJamGroupId = -1;
      }
   }
   else {//bIsInJam == false
      [iV + iVChange == 0] {
         bIsInJam = true;
         [diff(myCar.iPosition, carInFront.iPosition) <= 1
            && do_test_jam( carInFront/Id ) == true] {
            //FALL B
            iJamGroupId = carInFront.iJamGroupId;
            [iJamGroupId < 0] {
               //SELTENER FALL: all ar in same JAM
               iJamGroupId = gaugings.iTotalTrafficJams;
               gaugings.iTotalTrafficJams++;
            }
         }
         else {
            //FALL A
            iJamGroupId = gaugings.iTotalTrafficJams;
            gaugings.iTotalTrafficJams++;
         }
      }
      else [diff(myCar.iPosition, carInFront.iPosition) <= 1] {
         [ do_test_jam( carInFront/Id ) == true] {
            //FALL C
            bIsInJam = true;
            iJamGroupId = carInFront.iJamGroupId;
         }
      }
   }
   return bIsInJam;
```

Algorithmus STEP1+STEP6+AUTO1+AUTOX

```
#define FUNC_COUNT 18

int iOpt = OP_DEFAULT;
int(*aFunc[FUNC_COUNT])(struct naschmodell *) = {
   [ 0] do newtick,
   [ 1] [settings.eMode == step1] {print newtick},
   [ 2] userinteractionSTEP1,
   [ 3] do accelerate,
   [ 4] [settings.eMode == step1] {print accelerate},
   [ 5] userinteractionSTEP1,
   [ 6] do retard,
   [ 7] [settings.eMode == step1] {print retard},
   [ 8] userinteractionSTEP1,
   [ 9] do dilly_dally,
   [10] [settings.eMode == step1] {print dilly_dally},
   [11] userinteractionSTEP1,
   [12] do drive,
   [13] [settings.eMode == step1] {print drive},
   [14] userinteractionSTEP1,
   [15] do test_jam(int iCarId),
   [16] [settings.eMode != AUTOX] {print test_jam}  print gaugings,
   [17] userinteraction
};

for (int iF = 0; iOpt != OP_EXIT && iOpt != OP_STOP && iOpt != OP_PRINT; iF++, iF %= FUNC_COUNT) {
   iOpt = aFunc[iF](pModell);
   if (iF == FUNC_COUNT - 1 && pModell->sGaugings.iTicks >= MAXTICKS) {
      iOpt = OP_STOP;
   }
}

return iOpt;
```

userinteraction

```
int iOpt = OP_DEFAULT;

[pModell->sSettings.eMode == step1 || pModell->sSettings.eMode == step6]
{
   do
   {
      iOpt = _getch();
   }
   while (iOpt != OP_EXIT
       && iOpt != OP_STOP
       && iOpt != OP_STEP
       && (pModell->sSettings.eTSaveToFile == off || iOpt != OP_PRINT));
}
else
{
   [ _kbhit() == 1 ]
   {
      iOpt = _getch();

      [iOpt != OP_EXIT
       && iOpt != OP_STOP
       && (pModell->sSettings.eTSaveToFile == off || iOpt != OP_PRINT)]
      {
         iOpt = OP_DEFAULT;
      }
   }
}

return iOpt;
```

userinteractionSTEP1

```
int iOpt = OP_DEFAULT;

[pModell->sSettings.eMode == step1]
{
   do
   {
      iOpt = _getch();
   }
   while (iOpt != OP_EXIT
       && iOpt != OP_STOP
       && iOpt != OP_STEP
       && (pModell->sSettings.eTSaveToFile == off || iOpt != OP_PRINT));
}

return iOpt;
```

Umwandlung durch Farbschemas

Ausgehend von den komprimierten Zuständen:

```
struct saveState ppsState[990][MAXTICKS];

struct saveState
{
   bool bIsInJam;
   int iVTotal;
   int iJamGroup;
   int iPosition;
};
```

Wird eine ***.bmp** Datei der größe **STREET_LENGTH x MAXTICKS** erzeugt.

Zur Umwandlung stehen folgende **Farbschemas** zur Verfügung:

F1 - (simple)

```
Street:         black
Car(bIsInJam):  red
Car(!bIsInJam): white
```

F2 - (depending !IsInJam:VTotal)

```
Street:         black
Car(bIsInJam):  red
Car(!bIsInJam): [iVTotal]{VMAX:GREEN,..,5:Mixed(GREEN,RED),..,0:RED}
```

F3 – (depending IsInJam:JamGroup)

```
Street:         black
Car(bIsInJam):  [iJamGroup%32]{32Colors}
Car(!bIsInJam): white
```

COLORMAP.lib

```
struct colormap
{
   int iWidth;
   int iHeight;
   unsigned char ** ppucRED;
   unsigned char ** ppucGREEN;
   unsigned char ** ppucBLUE;
};

union uCOLOR
{
   unsigned int uiHEX;
   struct
   {
      unsigned char ucBLUE;
      unsigned char ucGREEN;
      unsigned char ucRED;
   } sRGB;
};

typedef const union uCOLOR * const COLOR;
typedef struct colormap * COLORMAP;

const union uCOLOR VALUE_RED = {.uiHEX = 0xff0000};
const union uCOLOR VALUE_GREEN = {.uiHEX = 0x00ff00};
const union uCOLOR VALUE_BLUE = {.uiHEX = 0x0000ff};
//usw..

#define RED   (&VALUE_RED)
#define GREEN (&VALUE_GREEN)
#define BLUE  (&VALUE_BLUE)
//usw..
```

Erweiterungen:

- **write log of settings + gaugings**

- **Kommandozeilenparameter**
  -#80 -vm20 -p33 -mQUIET -cON -idv50 -f123“test“

  -#80: 80 Autos.
  -vm20: Vmax = 20.
  -p33: P-DD = 33.
  -mQUIET: keine Ausgaben,
  nach der Simulation werden die Werte von Settings+Gaugings über stdout ausgegeben
  und nicht in ein logfile UND
  Programm beendet sich selbständig.
  -cON: Cruise-control = ON.
  -idv50: Increased delay at v=0: 50 (ON).
  -f123“test“: Save to file = ON, Farbschema = [F1,F2,F3], Filepath = “./test.bmp“.

- **solve iJamGroupId F3 BUG**