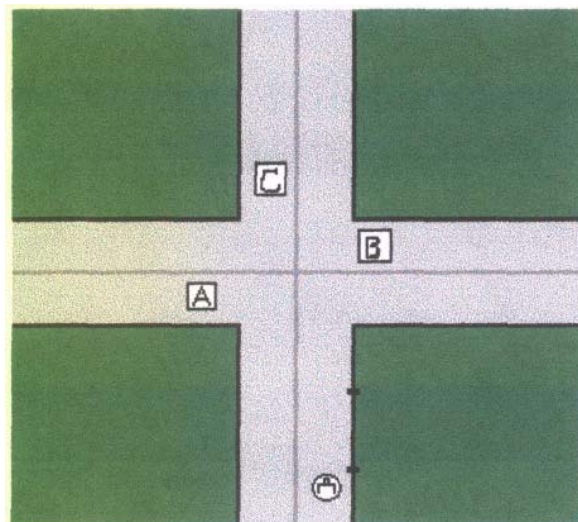


Задача – равносторонний перекресток.

Задача:

Дан равносторонний перекресток (общий случай) -



Наша машинка в виде маленького круга, то есть где мы находимся, все остальные машинки в виде квадратиков.

Теперь опишем правила :

В принципе по правилам дорожного движения всегда пропускаем помеху справа, но на нерегулируемом перекрестке в некоторых случаях оно применяется, в некоторых нет поэтому жестко установим ограничения.

- Если автомобиль Д то есть мы едем прямо, то пропускаем автомобиль В, если такой на перекрестке есть.
- Если автомобиль Д поворачивает налево то пропускаем автомобили С и В.
- Если автомобиль Д поворачивает направо, то никого не пропускает.

У меня получилось две таблицы :

- первая отвечает за приближение автомобиля Д к перекрестку.
- вторая отвечает за правила движения на перекрестке.

Первая таблица:

The screenshot shows the SIMP 2000 beta software window. It contains a logic table with 5 columns (1, 2, 3, 4, E) and 10 rows. The first 3 rows contain truth values (T, F), the next 4 rows contain binary values (1), and the last row contains a sum (+). To the right of the table is a rule editor with two rules: C1 -> IC2&IC3 and C2 -> IC3.

	1	2	3	4	E
1	T			F	
2		T		F	
3			T	F	
1	1				
2		1			
3			1		
4					
+	2	2	2	2	

Rules:

- C1 -> IC2&IC3
- C2 -> IC3

Опишем правила и действия таблицы 1 :

The screenshot shows the 'Названия условий и действий' dialog box. It has two sections: 'Условия' (Conditions) and 'Действия' (Actions). The 'Условия' section lists C1, C2, and C3 with their corresponding descriptions. The 'Действия' section lists A2, A3, and A4 with their corresponding descriptions. At the bottom are 'Да' (Yes) and 'Нет' (No) buttons.

Условия	
C1	Машина Д далеко от перекрестка
C2	Машина Д близко от перекрестка
C3	Машина Д на перекрестке

Действия	
A2	Тормозить
A3	Остановка
A4	Пустое действие

Buttons: Да, Нет

Таблица вторая:

The screenshot shows the SIMP 2000 Beta software interface. The main window displays a logic table with 8 columns (1-8) and 6 rows (1-6). The table contains truth values (T, F) and numbers (1, 2, 3). To the right of the table is a rule editor with the following rules:

```

C1 -> !C2 & !C3
C2 -> !C3
  
```

	1	2	3	4	5	6	7	8	E
1	T							F	
2		T	T					F	
3				T	T	T	T	F	
4		T	F	T	T	F	F		
5				T	F	T	F		
6									
+	S	S	S	S	S	S	S	1	

Опишем правила и действия таблицы 2 :

The screenshot shows the 'Названия условий и действий' (Names of conditions and actions) dialog box. It contains two sections: 'Условия' (Conditions) and 'Действия' (Actions).

Условия	
C1	Автомобиль Д хочет ехать направо
C2	Автомобиль Д хочет ехать прямо
C3	Автомобиль Д хочет ехать налево
C4	Справа есть машина

Действия	
R1	Д едет направо
R2	Д едет прямо
R3	Д едет налево
R4	Д ждет пока проедет авто, который справа

Buttons: Да (Yes), Нет (No)

Условия и действия, которые не влезли : C5 -
Есть навстречу машина.

R5 - Д ждет пока проедет авто, который навстречу. R6 -
Пустое действие.

Теперь давайте проверим таблицы на полноту и не противоречивость.

Таблица1:

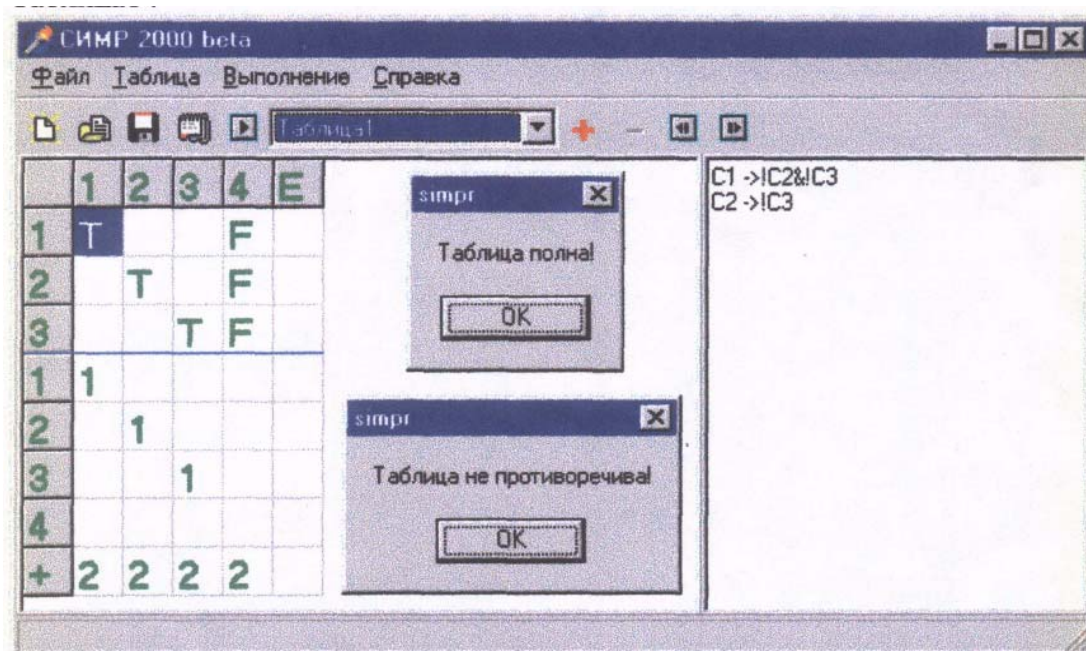
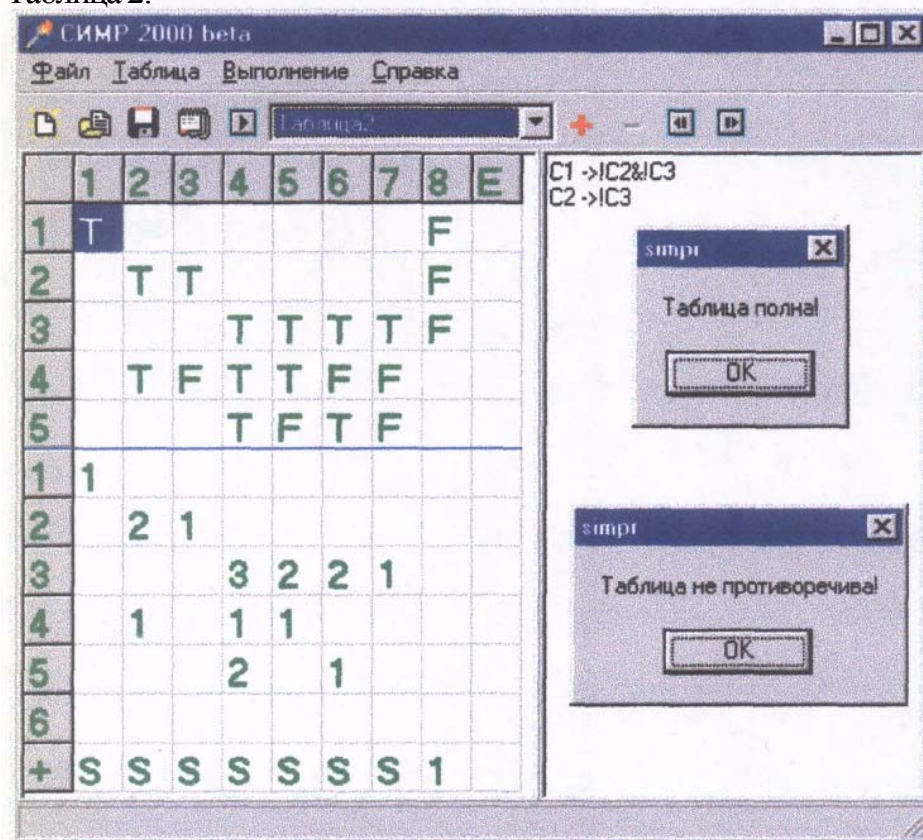
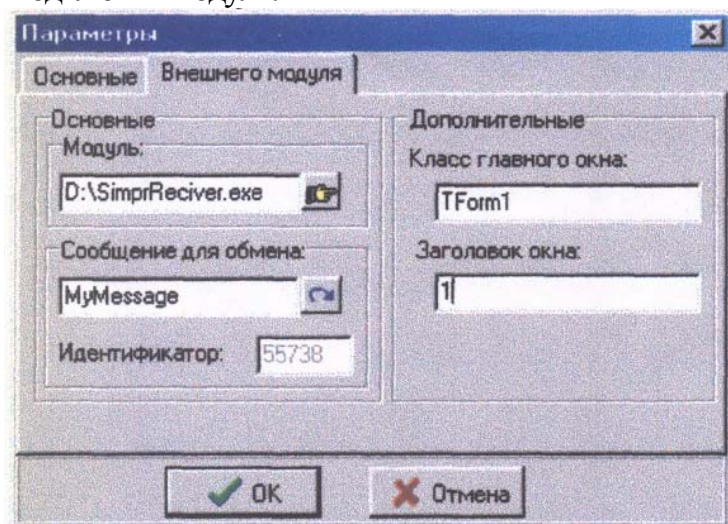


Таблица 2:



Подключим модуль:

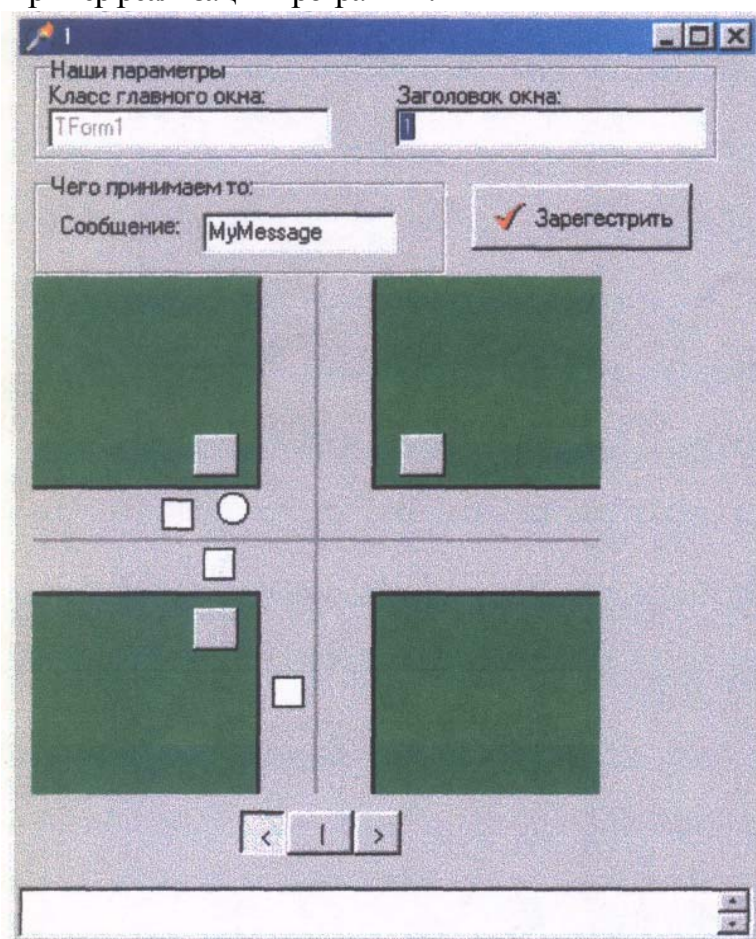


The 'Parameters' dialog box has two tabs: 'Basic' and 'External module'. The 'External module' tab is active. It contains the following fields:

- Basic:**
 - Module: with a file explorer icon.
 - Message for exchange: with a message icon.
 - Identifier:
- Additional:**
 - Main window class:
 - Window title:

Buttons at the bottom: and (Cancel).

Пример реализации программы:



The 'Our parameters' dialog box contains the following elements:

- Our parameters:**
 - Main window class:
 - Window title:
- What we receive:**
 - Message:
 - ☒ Зарегистрировать (Registered)
- Visual representation:** A 2x2 grid of green squares representing a crossroads. Each square contains a small grey car icon. Below the grid are three buttons: , , and , which control the car's direction.
- Status bar:** A white bar at the bottom with a small icon on the right.

Три кнопочки внизу задают направление автомобиля, кнопочки по углам задают, есть ли машины на перекрестке.

Ниже приведем текст модуля программы (написано на Delphi):

```
unit uLogo;
interface    uses Windows, Classes, SysUtils, Math, Dialogs,
Graphics; type
  TRoad = class(TObject)
    Stop: boolean;
    C1, C2, C3, C4, C5, C6: boolean;
    Bmp: TBitmap;
    Canvas: TCanvas;
    CarPos: TPoint;
    OtherPos: array[1..3] of TPoint; {right, forward, left}
    z1, z2: boolean;
    function CanMove: boolean;
    constructor Create;
    procedure Draw;
  procedure DrawCar;
    function CC1: boolean;
    function CC2: boolean;
    function CC3: boolean;
    procedure AA1;
    procedure AA2;
    procedure AA3;
    procedure A1;
    procedure A2;
    procedure A3;
    procedure A4;
    procedure A5;
    procedure A6;
  end;

  TMoveThread = class (TThread)
    Road: TRoad;
    Count: integer;
    Index: integer;
    procedure Dolt; virtual; abstract;
    procedure Execute; override;
    constructor Create (Owner: TRoad);
    property Terminated;
  end;

  TCarMove = class(TMoveThread)
    procedure Execute; override;
  end;

  TCarForward = class (TCarMove)
    procedure Dolt; override;
  end;

  TCarRotateLeft = class (TCarMove)
    Alpha: extended;
    constructor Create (Owner: TRoad);
    procedure Dolt; override;
  end;

  TCarRotateRight = class (TCarMove)
    Alpha: extended;
    constructor Create (Owner: TRoad);
    procedure Dolt; override;
  end;
```

```

TRightCarDraw = class (TMoveThread)
    procedure Execute; override;
    procedure Dolt; override;
end;

TForwardCarDraw = class (TMoveThread)
    procedure Execute; override;
    procedure Dolt; override;
end;

procedure Delay(ms: Cardinal);

implementation

uses RecForm;

function RotateArround(P: TPoint; Center: TPoint; Alpha: extended):
TPoint;
var x, y: extended;
    nx, ny: extended;
begin
    x:=p.x-center.x;
    y:=p.y-center.y;
    nx:=x*cos (Alpha) - y*sin (Alpha) ;
    ny:=x*sin (Alpha) + y*cos (Alpha) ;
    result .X:=Round (nx) + Center.X;
    result.Y:=Round(ny) + Center.Y;
end;

constructor TCarRotateLeft.Create(Owner: TRoad); begin
    inherited;
    Alpha:=0;
end;

procedure TCarRotateLeft.Dolt; begin
    Delay(100);
    Road.CarPos:=RotateArround(Road.CarPos, Point(94, 180), -pi/80);
    Form1.DrawAll;
end;

constructor TCarRotateRight.Create(Owner: TRoad);
begin
    inherited;
    Alpha :=0;
end;

procedure TCarRotateRight.Dolt;
begin
    Delay(100);
    Road.CarPos:=RotateArround(Road.CarPos, Point(194, 179), pi/80);
    Form1.DrawAll;
end;

procedure TCarForward.DoIt;
begin
    Delay(100);
    dec (Road.CarPos.Y, 3);
    Form1.DrawAll;
end;

```

```

procedure TRightCarDraw.Execute;
begin
    Road.z1:=false;
    while not Road.z2 do
    begin
        end;
        inherited;
        Road.z1 :=true;
    end;

procedure TRightCarDraw.DoIt;
begin
    Delay(100) ;
    dec(Road.OtherPos [1] .x, 3);
    Form1.DrawAll;
end;

procedure TForwardCarDraw.Execute;
begin
    Road.z2:=false;
    while not Road.z1 do
    begin
        end;
        inherited;
        Road.z2:=true;
    end;

procedure TForwardCarDraw.DoIt;
begin
    Delay(100);
    inc(Road.OtherPos[2] .y, 3);
    Form1.DrawAll;
end;

procedure TCarMove. Execute;
begin
    while not Road.CanMove do
    begin
        end;
        inherited;
    end;

procedure TMoveThread.Execute;
var i: integer;
begin
    for i:=0 to Count-1 do
    begin
        Index:=i;
        Synchronize (DoIt) ;
    end;
end;

constructor TMoveThread.Create (Owner: TRoad);
begin
    inherited Create (false);
    FreeOnTerminate:=true;
    Count:=40;
    Road:=Owner;
end;

```



```

function TRoad.CanMove: boolean;
begin
    result:=z1 and z2;
end;

procedure TRoad.A1;
var thr: TMoveThread;
begin
    thr:=TCarRotateRight.Create(self);
end
procedure TRoad.A2;
var thr: TMoveThread;
begin
    thr:=TCarForward.Create (self);
end;

procedure TRoad.A3;
var thr: TMoveThread;
begin
    thr :=TCarRotateLeft.Create (self);
end;

procedure TRoad.A4;
var thr: TMoveThread;
begin
    thr:=TRightCarDraw.Create(self);
end;

procedure TRoad.A5;
var thr: TMoveThread;
begin
    thr :=TForwardCarDraw.Create (self);
end;

procedure TRoad.A6;
begin
end;

procedure TRoad.AA1;
begin
    if Stop then exit;
    dec(CarPos.Y, 2);
    Form1.DrawAll;
end;

procedure TRoad.AA2;
begin
    if Stop then exit;
    dec (CarPos.Y, 1);
    Form1.DrawAll;
end;

procedure TRoad.AA3;
begin
    Stop:=true;
    Form1.DrawAll;
end;

function TRoad.CC1: boolean;
begin

```

```

    result:= (CarPos.Y<=256) and (CarPos.Y>213)
end;

function TRoad.CC2: boolean;
begin
    result:=(CarPos.Y<=213) and (CarPos.Y>179); end;

function TRoad.CC3: boolean;
begin
    result:=(CarPos.Y<=179)
end;

procedure TRoad.DrawCar;
begin
    Canvas.Brush.Color:=clWhite;
    Canvas.Pen.Color:=clBlack;
    with CarPos do
        Canvas.Ellipse (x-8, y-8, x+8, y+8);
end;

procedure TRoad.Draw;
begin
    Canvas.Draw(0, 0, Bmp);
    Canvas.Brush.Color:=clWhite;
    Canvas.Pen.Color:=clBlack;
    if C4 then

        ;
        begin
            with OtherPos[1] do
                Canvas.Rectangle(x-8, y-8, x+8, y+8) ;
            end;
        if C5 then
            begin
                with OtherPos[2] do
                    Canvas.Rectangle(x-8, y-8, x+8, y+8);
                end;
            if C6 then
                begin
                    with OtherPos[3] do
                        Canvas.Rectangle(x-8, y-8, x+8, y+8) ;
                    end;
                end;
            end;
        end;

constructor TRoad.Create;
begin
    inherited;
    C4:=false; C5:=false; C6:=false;
    C1:=false; C2:=true; C3:=false;
    Bmp:=TBitmap.Create;
    Bmp.LoadFromFile('Croad.bmp');
    Canvas:=nil;
    CarPos:=Point( 156, 256);
    Stop:=false;
    OtherPos[1]:=Point(193, 121);
    OtherPos[2]:=Point(129, 91);
    OtherPos[3]:=Point(94, 146);
    z1:=true; z2:=true;
end;

procedure Delay(ms: Cardinal);
var l: Cardinal;

```

```
begin
  l:=GetTickCount; while
GetTickCount-l<ms do
  begin
  end;
```