**Proiect: Depășire**

### Conceptul proiectului

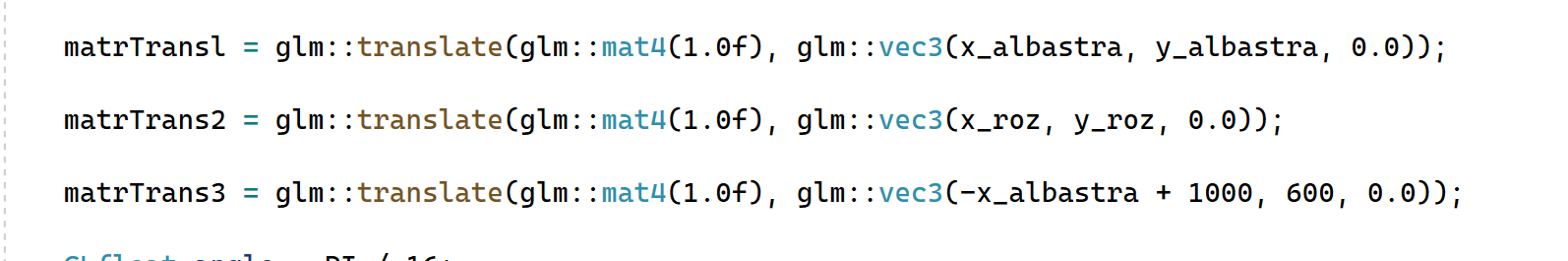
Acest proiect prezintă depășirea din trafic, conceptul fiind acela că există trei mașini, doua care circulă din sensuri opuse și o a treia care efectuează depășirea

Mașinile au elemente de design precum faruri, iar cea care depășește semnalizează. Sensurile de mers sunt și ele despărțite de un marcaj discontinuu format din dreptunghiuri albe.

### Transformări

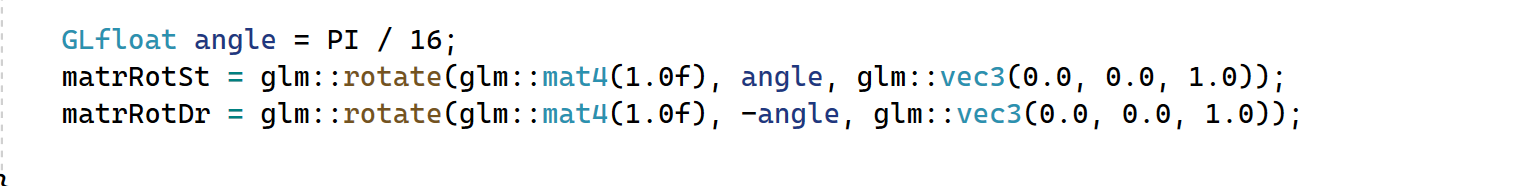
#### Translație

Am folosit translații pentru a deplasa cele trei mașini pe axe, cele două pe Ox și a treia pe Ox și Oy, deoarece depășește.



#### Rotație

Mașina care depășește își schimbă orientarea atunci când merge în sus sau în jos pe axa Oy, pentru un aspect mai realist.



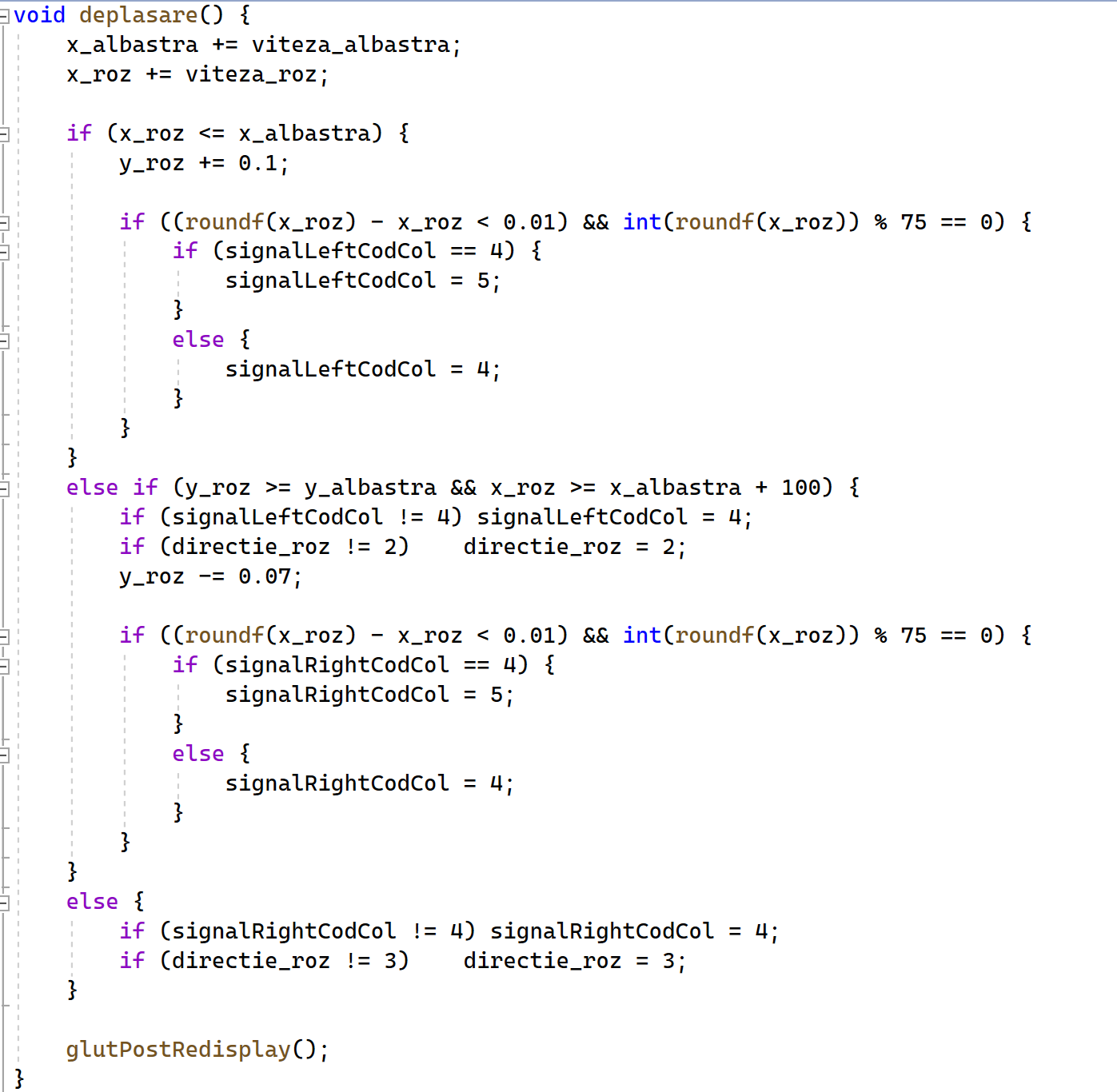
### 3. Originalitate

Proiectul nostru abordează o temă originală prin adăugarea elementelor de design cum ar fi farurile (și semnalizarea pe ambele direcții), dar și prin modul natural prin care se deplasează vehiculele.

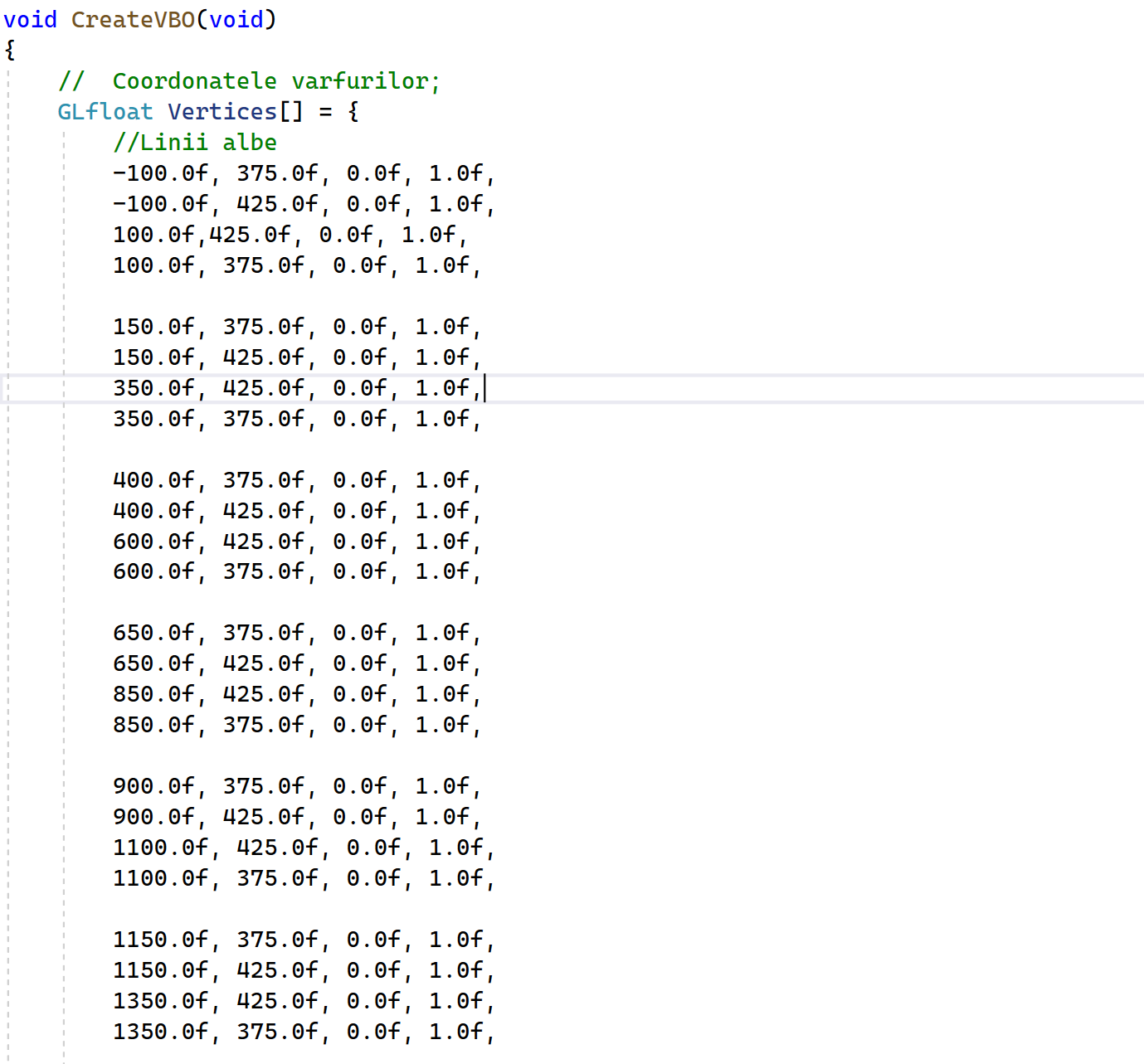
### 4. Capturi de ecran relevante

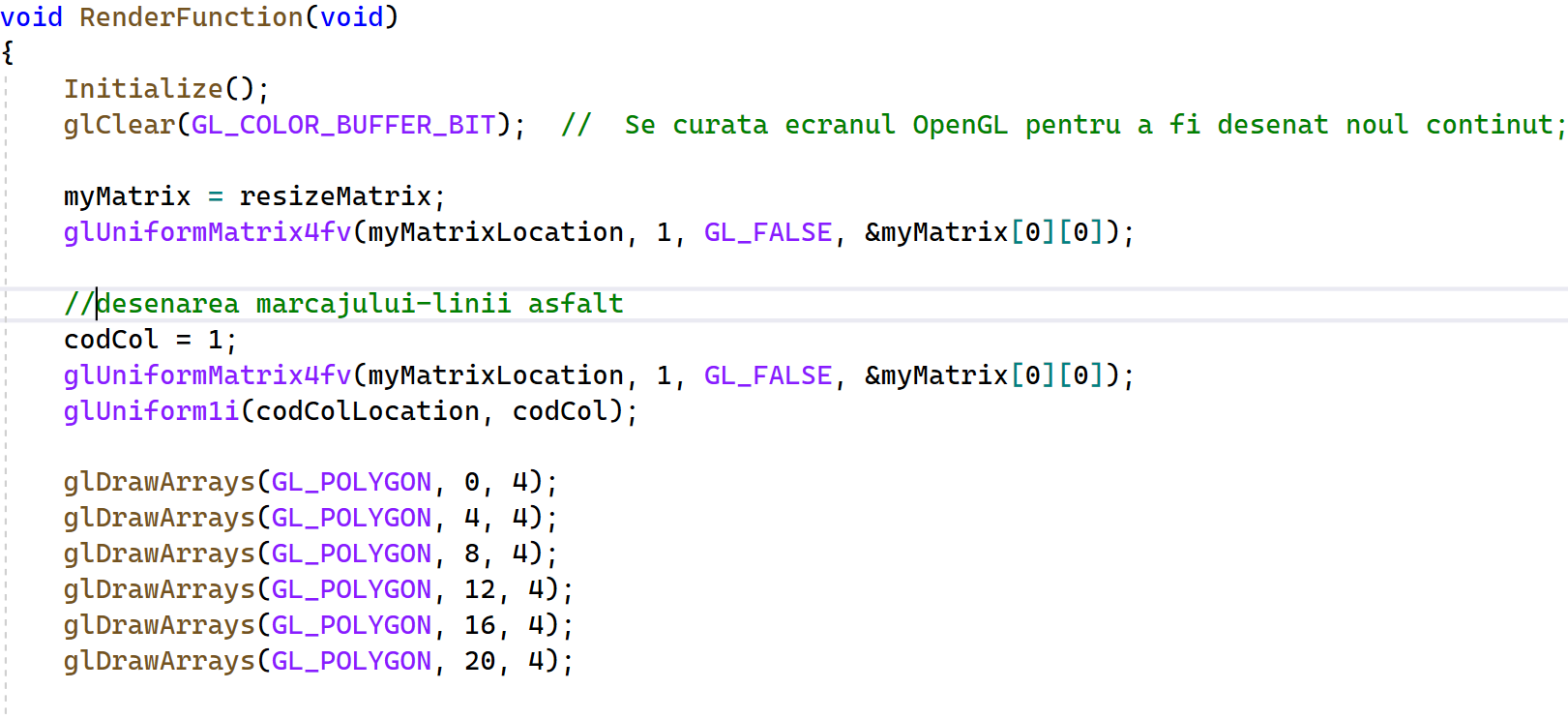
#### cod

* funcția de deplasare prin care mutăm pe axa Ox mașina care va fi depășită și în funcție de locația ei stabilim cum se mișcă cea care depășește; de asemenea, în funcție de urcarea sau coborârea pe axa Oy activăm semnalizarea.

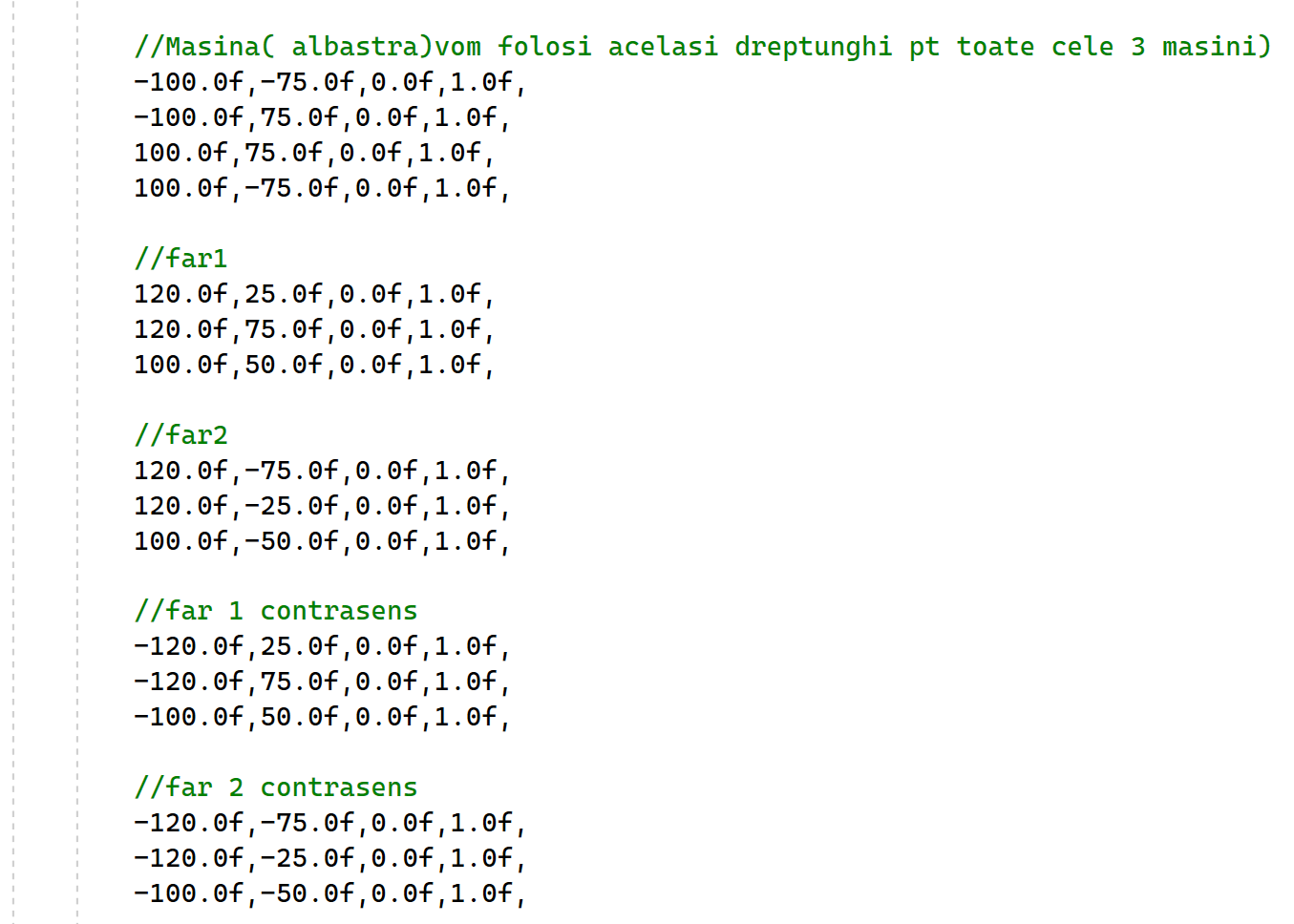


* trasarea marcajului care desparte benzile

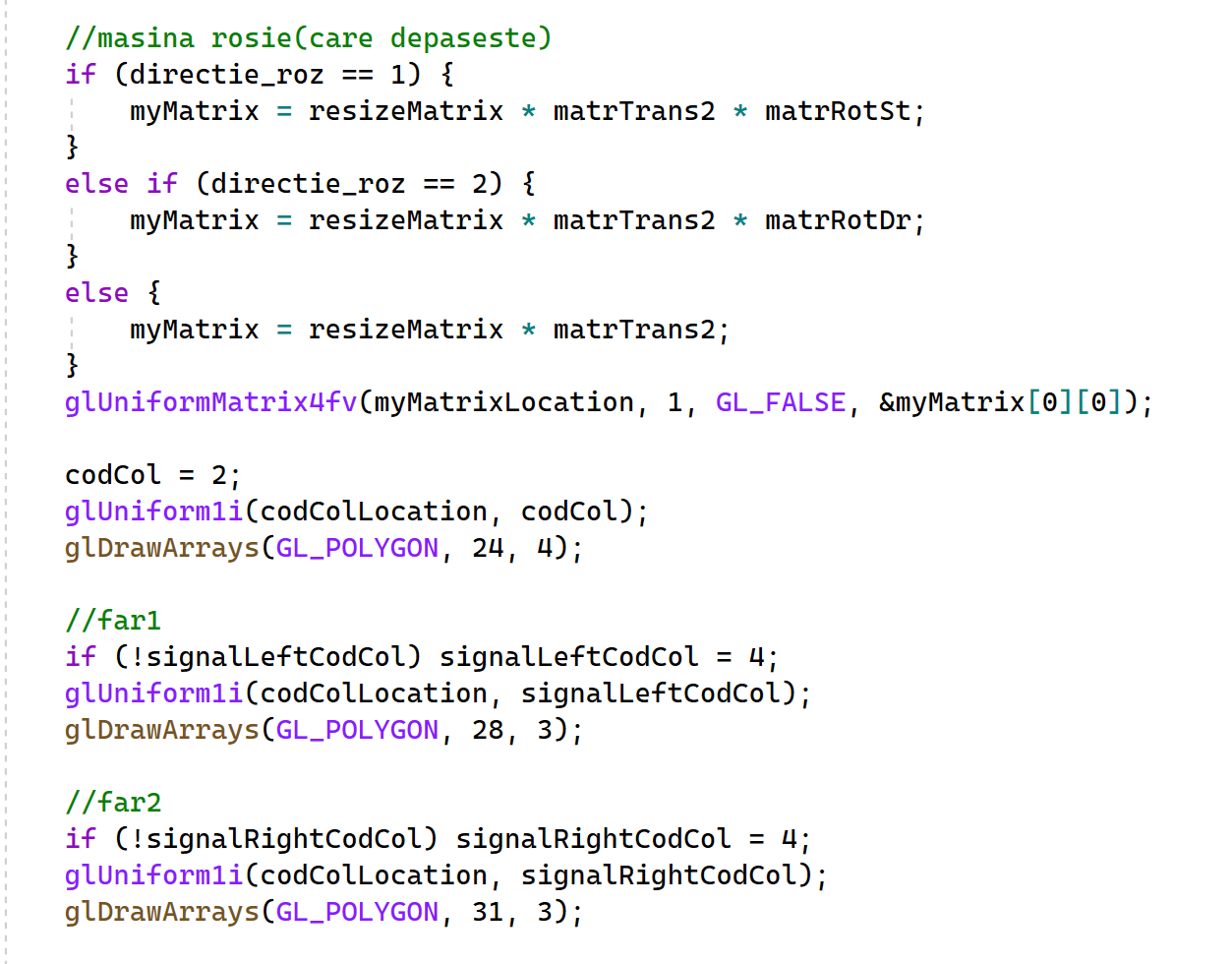


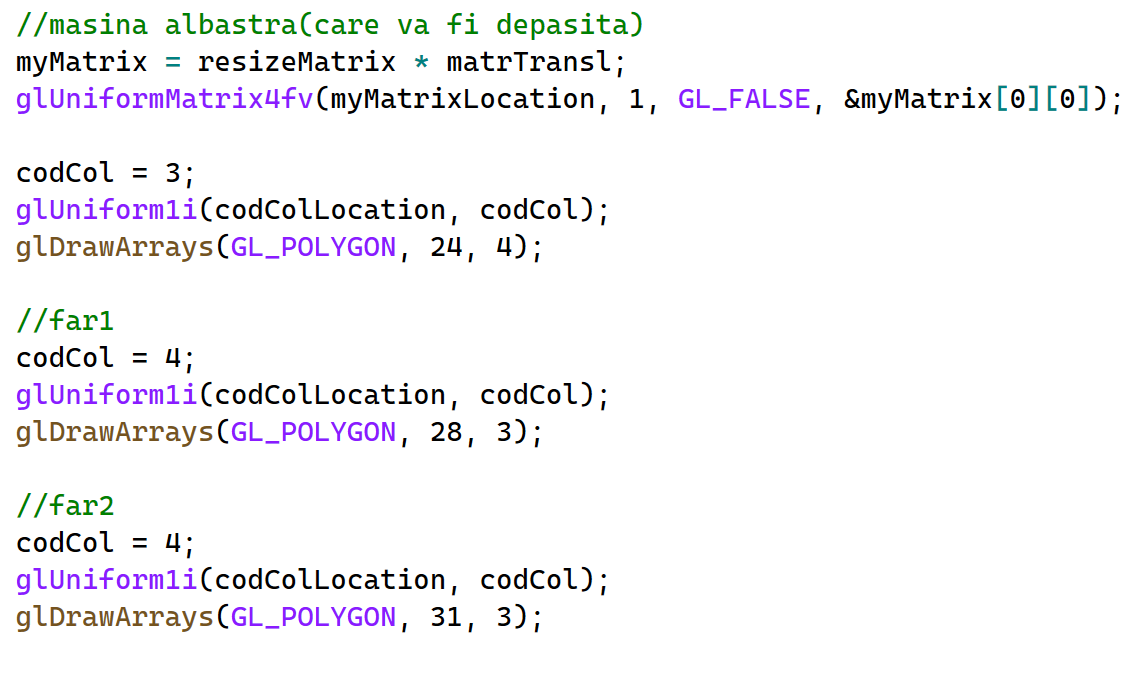


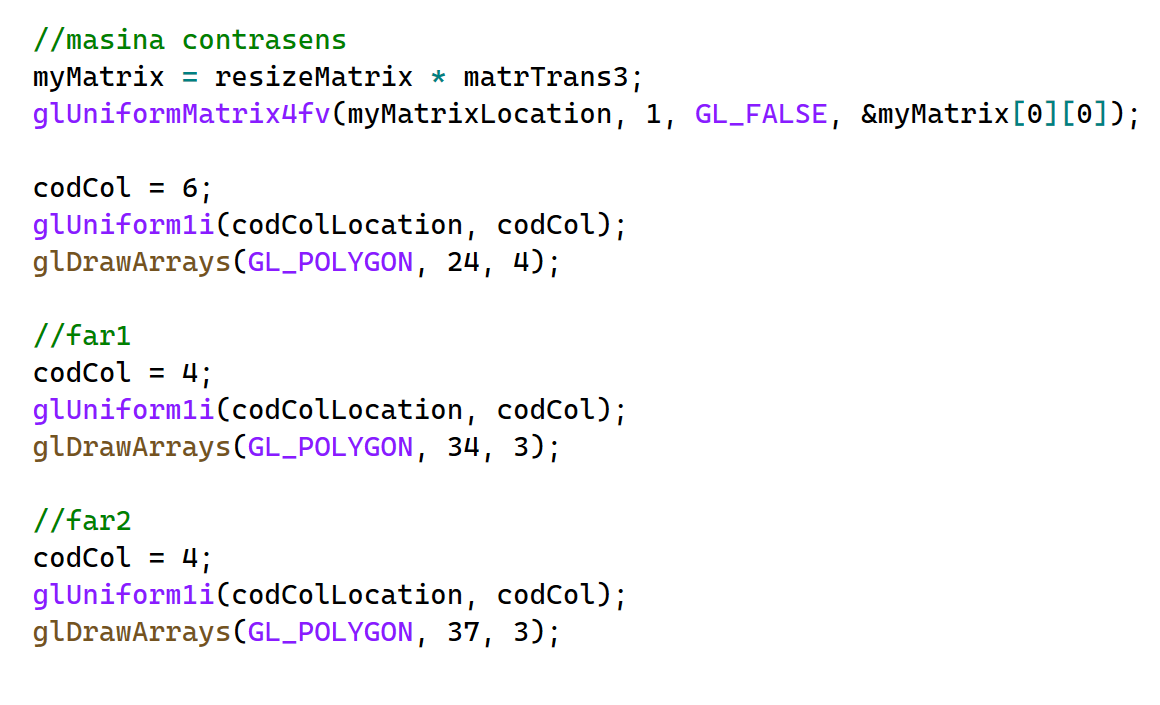
* desenarea dreptunghiului care va reprezenta cele trei mașini împreună cu farurile sale



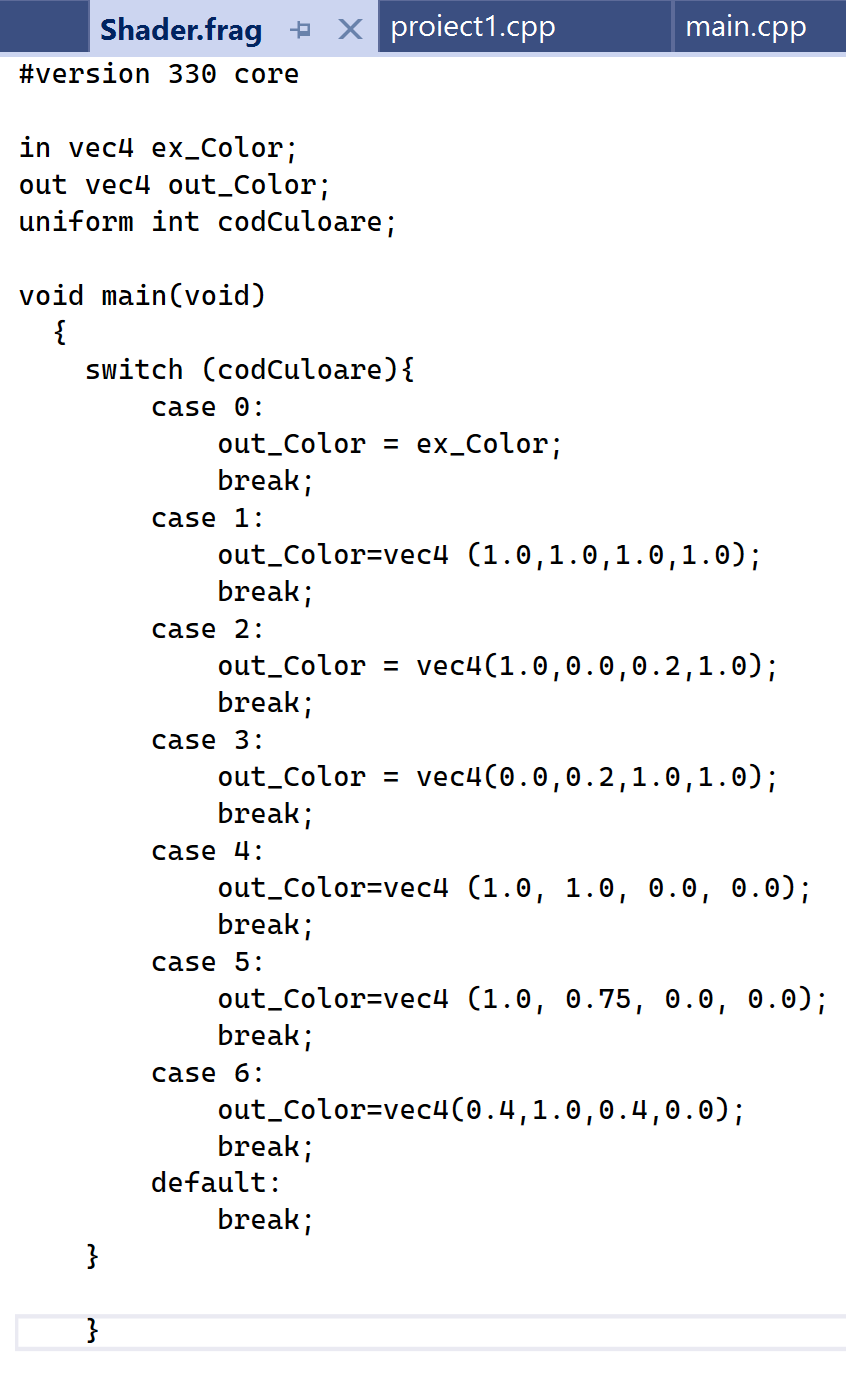
* plasarea celor trei mașini în scenă



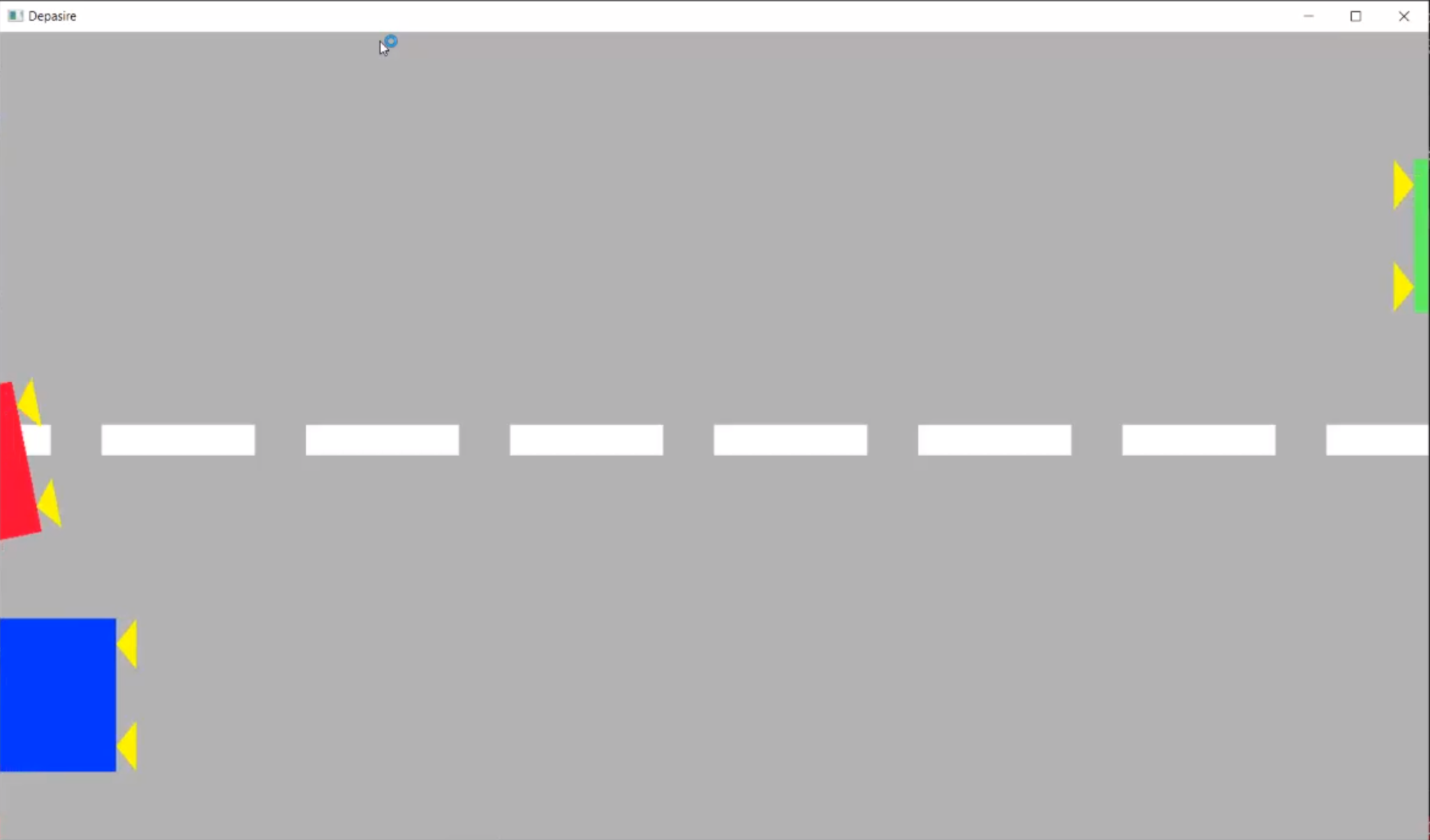


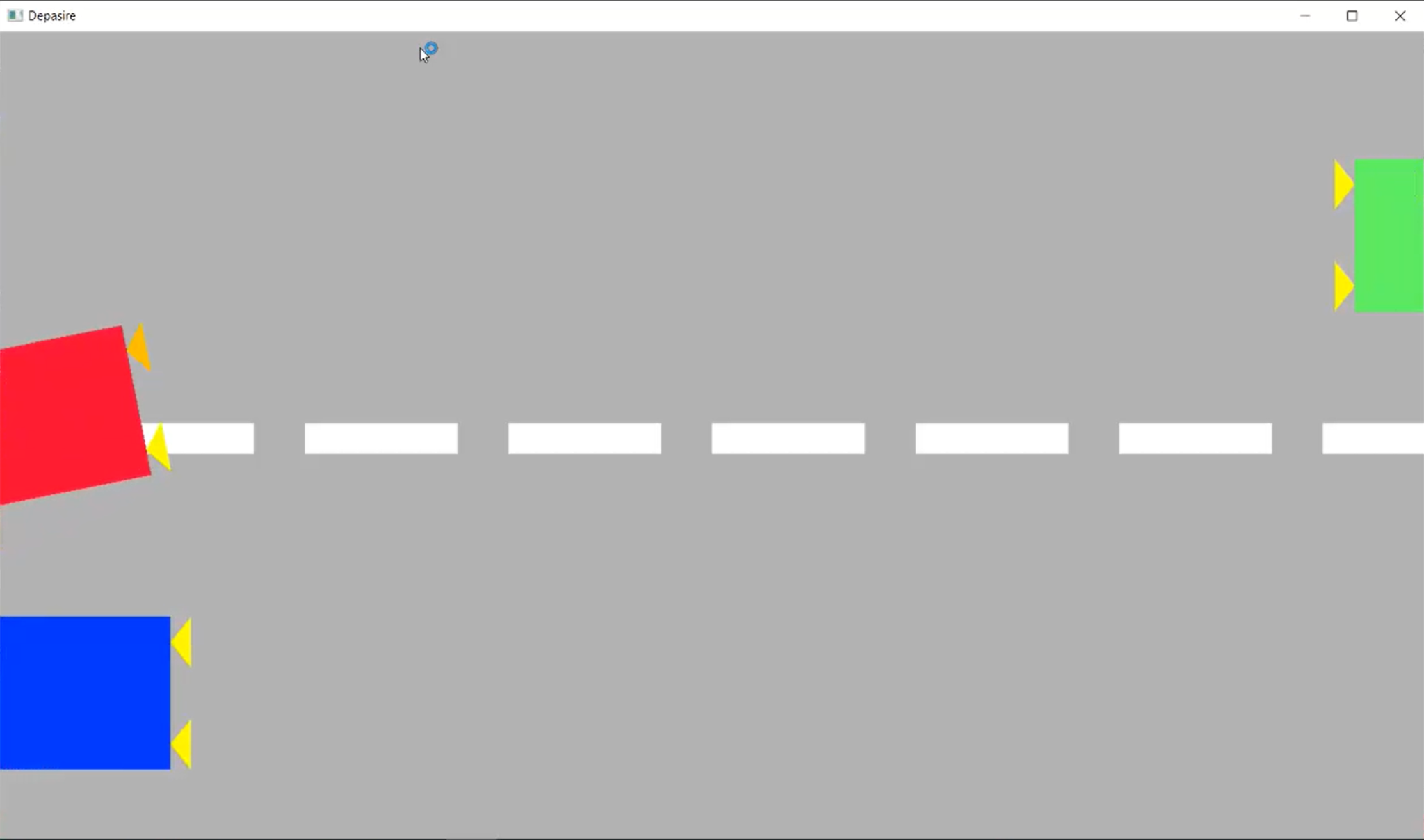


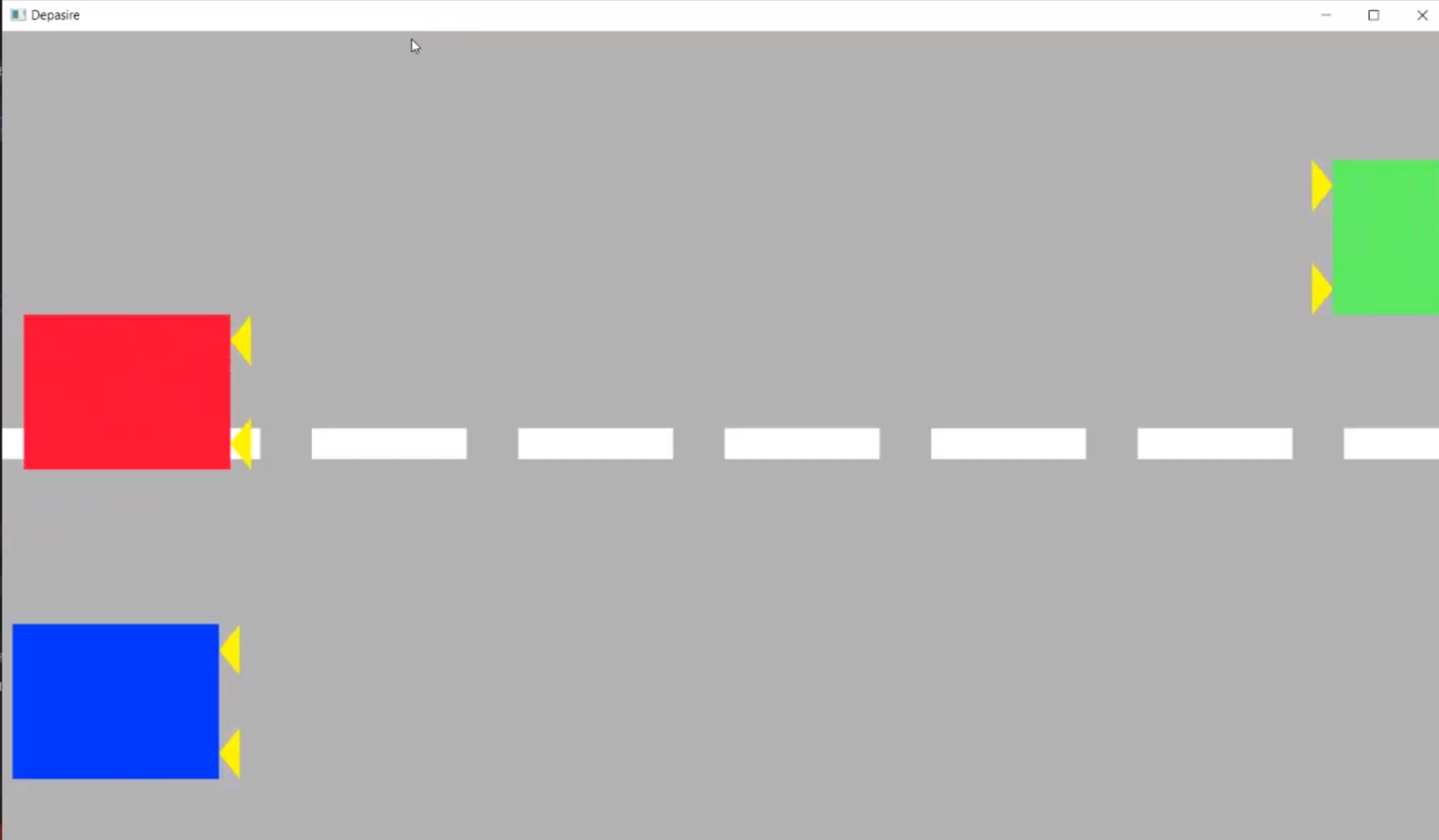
* culorile din Shader

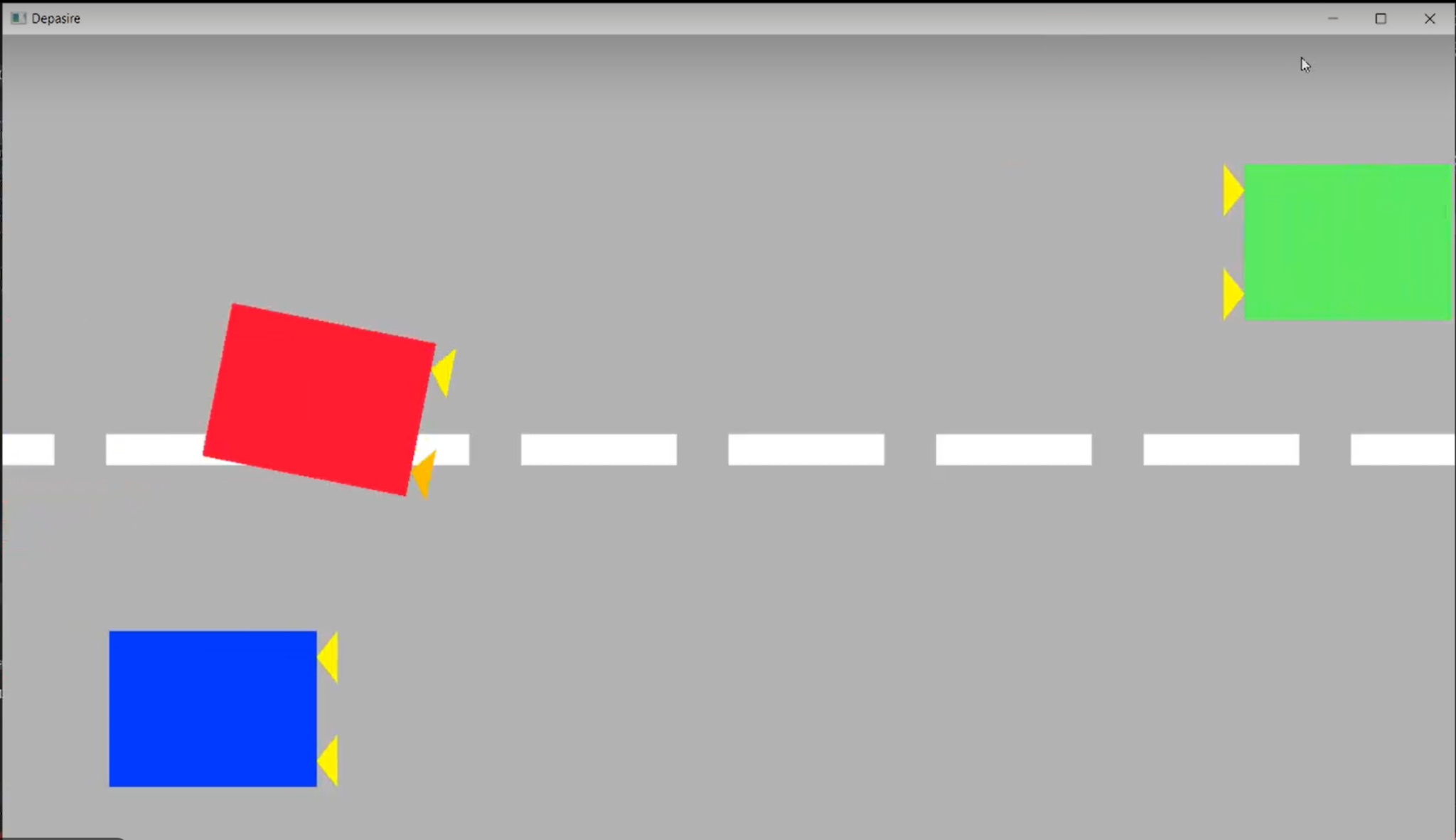


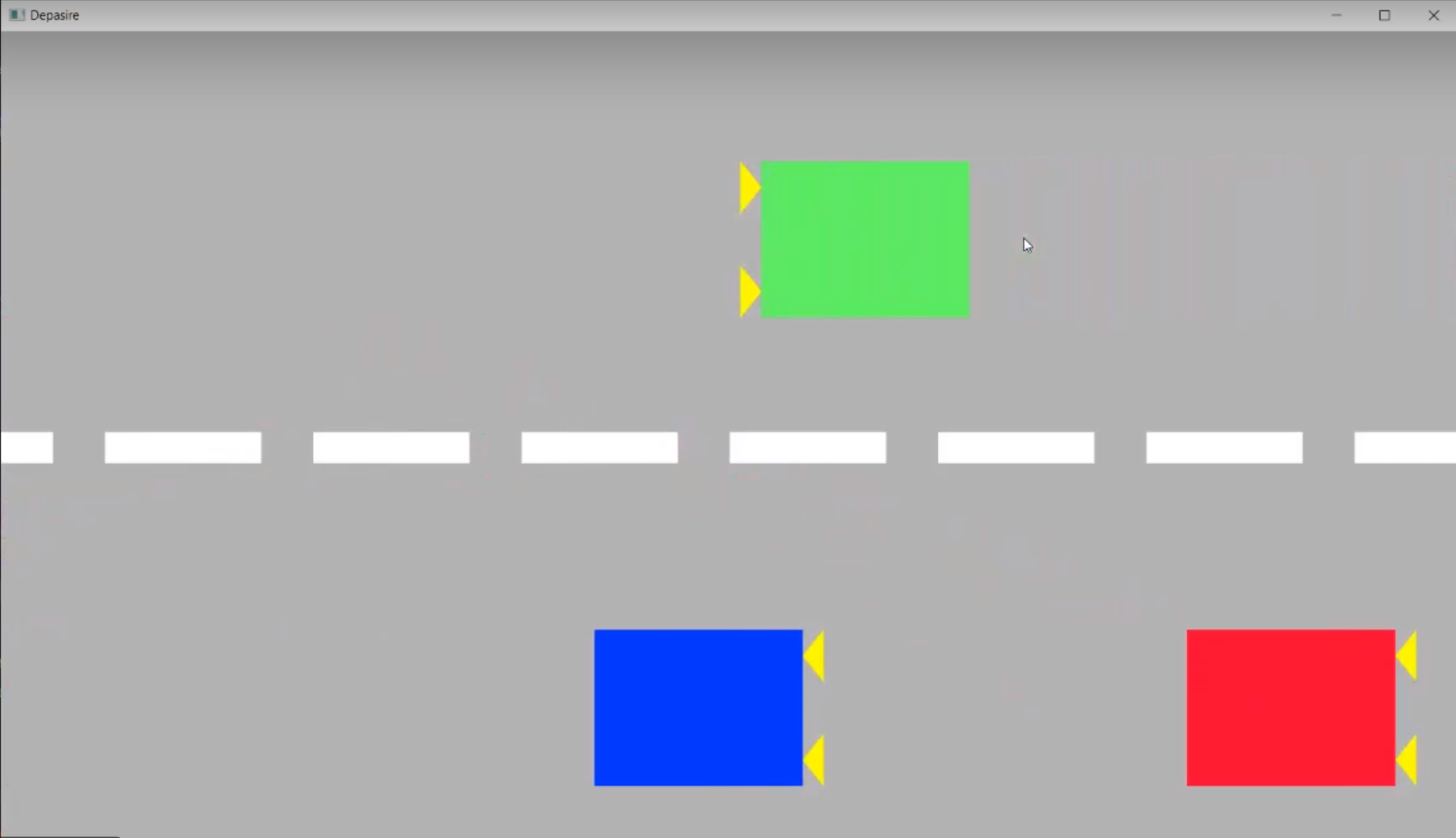
#### rulare











### 

### 5. Contribuții individuale

* Monete Maria
* rotația mașinii atunci când depășește
* desenarea mașinilor
* Oprea Mihai Ștefan
* mutarea mașinii roșii pe axa Oy pentru depășire
* semnalizarea mașinii roșii
* Oprea Tudor
* trasarea marcajului dintre benzi
* mișcarea pe axa Ox a tuturor mașinilor

### 6. Codul complet:

#### Main

// Codul sursa este adaptat dupa OpenGLBook.com

#include <windows.h> // biblioteci care urmeaza sa fie incluse

#include <stdlib.h> // necesare pentru citirea shader-elor

#include <stdio.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "loadShaders.h"

#include "glm/glm.hpp"

#include "glm/gtc/matrix\_transform.hpp"

#include "glm/gtx/transform.hpp"

#include "glm/gtc/type\_ptr.hpp"

//////////////////////////////////////

GLuint

VaoId,

VboId,

ColorBufferId,

ProgramId,

matrScaleLocation,

matrTranslLocation,

matrRotlLocation,

codColLocation,

myMatrixLocation;

int codCol, signalLeftCodCol, signalRightCodCol;

float width = 1400, heigth = 800;

float x\_albastra = -200.0, y\_albastra = 150.0, viteza\_albastra = 0.1;

float x\_roz = -500.0, y\_roz = 150.0, viteza\_roz = 0.2, directie\_roz = 1;

float PI = 3.141592;

glm::mat4 myMatrix, resizeMatrix, matrTransl, matrTrans2, matrTrans3, matrRotSt, matrRotDr;

void deplasare() {

// La fiecare apelare a funcției deplasăm mașina roșie și cea albastră spre dreapta

x\_albastra += viteza\_albastra;

x\_roz += viteza\_roz;

// Dacă mașina roșie se află în spatele albastrei, o urcăm pe axa Oy

if (x\_roz <= x\_albastra) {

y\_roz += 0.1;

// și o facem să semnalizeze stânga odată la un interval de timp

if (int(roundf(x\_roz)) % 75 == 0) {

if (signalLeftCodCol == 4) {

signalLeftCodCol = 5;

}

else {

signalLeftCodCol = 4;

}

}

}

// După ce a trecut de mașina albastră, cea roșie își schimbă orientarea

else if (y\_roz >= y\_albastra && x\_roz >= x\_albastra + 100) {

if (signalLeftCodCol != 4) signalLeftCodCol = 4;

if (directie\_roz != 2) directie\_roz = 2;

y\_roz -= 0.07;

//și dă semnal în dreapta

if (int(roundf(x\_roz)) % 75 == 0) {

if (signalRightCodCol == 4) {

signalRightCodCol = 5;

}

else {

signalRightCodCol = 4;

}

}

}

// Când ajung cele două paralele, cea roșie se îndreaptă în față

// la fel și după ce a depășit, când sunt ambele pe bandă

else {

if (signalRightCodCol != 4) signalRightCodCol = 4;

if (directie\_roz != 3) directie\_roz = 3;

}

glutPostRedisplay();

}

void CreateVBO(void)

{

// Coordonatele varfurilor;

GLfloat Vertices[] = {

//Linii albe

-100.0f, 385.0f, 0.0f, 1.0f,

-100.0f, 415.0f, 0.0f, 1.0f,

50.0f,415.0f, 0.0f, 1.0f,

50.0f, 385.0f, 0.0f, 1.0f,

100.0f, 385.0f, 0.0f, 1.0f,

100.0f,415.0f, 0.0f, 1.0f,

250.0f, 415.0f, 0.0f, 1.0f,

250.0f, 385.0f, 0.0f, 1.0f,

300.0f, 385.0f, 0.0f, 1.0f,

300.0f, 415.0f, 0.0f, 1.0f,

450.0f, 415.0f, 0.0f, 1.0f,

450.0f, 385.0f, 0.0f, 1.0f,

500.0f, 385.0f, 0.0f, 1.0f,

500.0f, 415.0f, 0.0f, 1.0f,

650.0f,415.0f, 0.0f, 1.0f,

650.0f, 385.0f, 0.0f, 1.0f,

700.0f, 385.0f, 0.0f, 1.0f,

700.0f, 415.0f, 0.0f, 1.0f,

850.0f, 415.0f, 0.0f, 1.0f,

850.0f, 385.0f, 0.0f, 1.0f,

900.0f,385.0f, 0.0f, 1.0f,

900.0f, 415.0f, 0.0f, 1.0f,

1050.0f, 415.0f, 0.0f, 1.0f,

1050.0f, 385.0f, 0.0f, 1.0f,

1100.0f,385.0f, 0.0f, 1.0f,

1100.0f, 415.0f, 0.0f, 1.0f,

1250.0f, 415.0f, 0.0f, 1.0f,

1250.0f, 385.0f, 0.0f, 1.0f,

1300.0f,385.0f, 0.0f, 1.0f,

1300.0f, 415.0f, 0.0f, 1.0f,

1450.0f, 415.0f, 0.0f, 1.0f,

1450.0f, 385.0f, 0.0f, 1.0f,

//Masina( albastra)vom folosi acelasi dreptunghi pt toate cele 3 masini)

-100.0f,-75.0f,0.0f,1.0f,

-100.0f,75.0f,0.0f,1.0f,

100.0f,75.0f,0.0f,1.0f,

100.0f,-75.0f,0.0f,1.0f,

//far1

120.0f,25.0f,0.0f,1.0f,

120.0f,75.0f,0.0f,1.0f,

100.0f,50.0f,0.0f,1.0f,

//far2

120.0f,-75.0f,0.0f,1.0f,

120.0f,-25.0f,0.0f,1.0f,

100.0f,-50.0f,0.0f,1.0f,

//far 1 contrasens

-120.0f,25.0f,0.0f,1.0f,

-120.0f,75.0f,0.0f,1.0f,

-100.0f,50.0f,0.0f,1.0f,

//far 2 contrasens

-120.0f,-75.0f,0.0f,1.0f,

-120.0f,-25.0f,0.0f,1.0f,

-100.0f,-50.0f,0.0f,1.0f,

};

GLfloat Colors[] = {

1.0f,1.0f,1.0f,0.0f,

};

// Culorile varfurilor din colturi;

glGenVertexArrays(1, &VaoId); // Generarea VAO si indexarea acestuia catre variabila VaoId;

glBindVertexArray(VaoId);

// Se creeaza un buffer pentru VARFURI;

glGenBuffers(1, &VboId); // Generarea bufferului si indexarea acestuia catre variabila VboId;

glBindBuffer(GL\_ARRAY\_BUFFER, VboId); // Setarea tipului de buffer - atributele varfurilor;

glBufferData(GL\_ARRAY\_BUFFER, sizeof(Vertices), Vertices, GL\_STATIC\_DRAW); // Punctele sunt "copiate" in bufferul curent;

// Se asociaza atributul (0 = coordonate) pentru shader;

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 4, GL\_FLOAT, GL\_FALSE, 0, 0);

// Se creeaza un buffer pentru CULOARE;

glGenBuffers(1, &ColorBufferId);

glBindBuffer(GL\_ARRAY\_BUFFER, ColorBufferId);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(Colors), Colors, GL\_STATIC\_DRAW);

// Se asociaza atributul (1 = culoare) pentru shader;

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 4, GL\_FLOAT, GL\_FALSE, 0, 0);

}

void DestroyVBO(void)

{

glDisableVertexAttribArray(1);

glDisableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glDeleteBuffers(1, &ColorBufferId);

glDeleteBuffers(1, &VboId);

glBindVertexArray(0);

glDeleteVertexArrays(1, &VaoId);

}

void CreateShaders(void)

{

ProgramId = LoadShaders("example.vert", "example.frag");

glUseProgram(ProgramId);

}

void DestroyShaders(void)

{

glDeleteProgram(ProgramId);

}

void Initialize(void)

{

glClearColor(0.7f, 0.7f, 0.7f, 0.7f); // Culoarea de fond a ecranului;

CreateVBO(); // Trecerea datelor de randare spre bufferul folosit de shadere;

CreateShaders(); // Initilizarea shaderelor;

// Instantierea variabilelor uniforme pentru a "comunica" cu shaderele;

codColLocation = glGetUniformLocation(ProgramId, "codCuloare");

myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");

resizeMatrix = glm::ortho(0.0f, width, 0.0f, heigth);

matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(x\_albastra, y\_albastra, 0.0));

matrTrans2 = glm::translate(glm::mat4(1.0f), glm::vec3(x\_roz, y\_roz, 0.0));

matrTrans3 = glm::translate(glm::mat4(1.0f), glm::vec3(-x\_albastra + 1500, 600, 0.0));

GLfloat angle = PI / 16;

matrRotSt = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0, 0.0, 1.0));

matrRotDr = glm::rotate(glm::mat4(1.0f), -angle, glm::vec3(0.0, 0.0, 1.0));

}

void RenderFunction(void)

{

Initialize();

glClear(GL\_COLOR\_BUFFER\_BIT); // Se curata ecranul OpenGL pentru a fi desenat noul continut;

myMatrix = resizeMatrix;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

//desenarea marcajului-linii asfalt

codCol = 1;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_POLYGON, 0, 4);

glDrawArrays(GL\_POLYGON, 4, 4);

glDrawArrays(GL\_POLYGON, 8, 4);

glDrawArrays(GL\_POLYGON, 12, 4);

glDrawArrays(GL\_POLYGON, 16, 4);

glDrawArrays(GL\_POLYGON, 20, 4);

glDrawArrays(GL\_POLYGON, 24, 4);

glDrawArrays(GL\_POLYGON, 28, 4);

//masina albastra(care va fi depasita)

myMatrix = resizeMatrix \* matrTransl;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

codCol = 3;

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_POLYGON, 32, 4);

//far1

codCol = 4;

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_POLYGON, 36, 3);

//far2

codCol = 4;

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_POLYGON, 39, 3);

//masina rosie(care depaseste)

if (directie\_roz == 1) {

myMatrix = resizeMatrix \* matrTrans2 \* matrRotSt;

}

else if (directie\_roz == 2) {

myMatrix = resizeMatrix \* matrTrans2 \* matrRotDr;

}

else {

myMatrix = resizeMatrix \* matrTrans2;

}

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

codCol = 2;

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_POLYGON, 32, 4);

//far1

if (!signalLeftCodCol) signalLeftCodCol = 4;

glUniform1i(codColLocation, signalLeftCodCol);

glDrawArrays(GL\_POLYGON, 36, 3);

//far2

if (!signalRightCodCol) signalRightCodCol = 4;

glUniform1i(codColLocation, signalRightCodCol);

glDrawArrays(GL\_POLYGON, 39, 3);

//masina contrasens

myMatrix = resizeMatrix \* matrTrans3;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

codCol = 6;

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_POLYGON, 32, 4);

//far1

codCol = 4;

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_POLYGON, 42, 3);

//far2

codCol = 4;

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_POLYGON, 45, 3);

glFlush(); // Asigura rularea tuturor comenzilor OpenGL apelate anterior;

}

void Cleanup(void)

{

DestroyShaders();

DestroyVBO();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 0); // pozitia initiala a ferestrei

glutInitWindowSize(1400, 800); //dimensiunile ferestrei

glutCreateWindow("Depasire"); // titlul ferestrei

glewInit(); // nu uitati de initializare glew; trebuie initializat inainte de a a initializa desenarea

Initialize();

glutDisplayFunc(RenderFunction);

glutIdleFunc(deplasare);

glutCloseFunc(Cleanup);

glutMainLoop();

}

#### Shader.vert

// Shader-ul de varfuri

#version 330

layout (location = 0) in vec4 in\_Position;

layout (location = 1) in vec4 in\_Color;

out vec4 gl\_Position;

out vec4 ex\_Color;

uniform mat4 myMatrix;

void main(void)

{

gl\_Position = myMatrix\*in\_Position;

ex\_Color = in\_Color;

}

#### Shader.frag

#version 330 core

in vec4 ex\_Color;

out vec4 out\_Color;

uniform int codCuloare;

void main(void)

{

switch (codCuloare){

case 0:

out\_Color = ex\_Color;

break;

case 1:

out\_Color=vec4 (1.0,1.0,1.0,1.0);

break;

case 2:

out\_Color = vec4(1.0,0.0,0.2,1.0);

break;

case 3:

out\_Color = vec4(0.0,0.2,1.0,1.0);

break;

case 4:

out\_Color=vec4 (1.0, 1.0, 0.0, 0.0);

break;

case 5:

out\_Color=vec4 (1.0, 0.75, 0.0, 0.0);

break;

case 6:

out\_Color=vec4(0.4,1.0,0.4,0.0);

break;

default:

break;

}

}