Documentatie Proiect Depasire

**Modificari adaugate:**

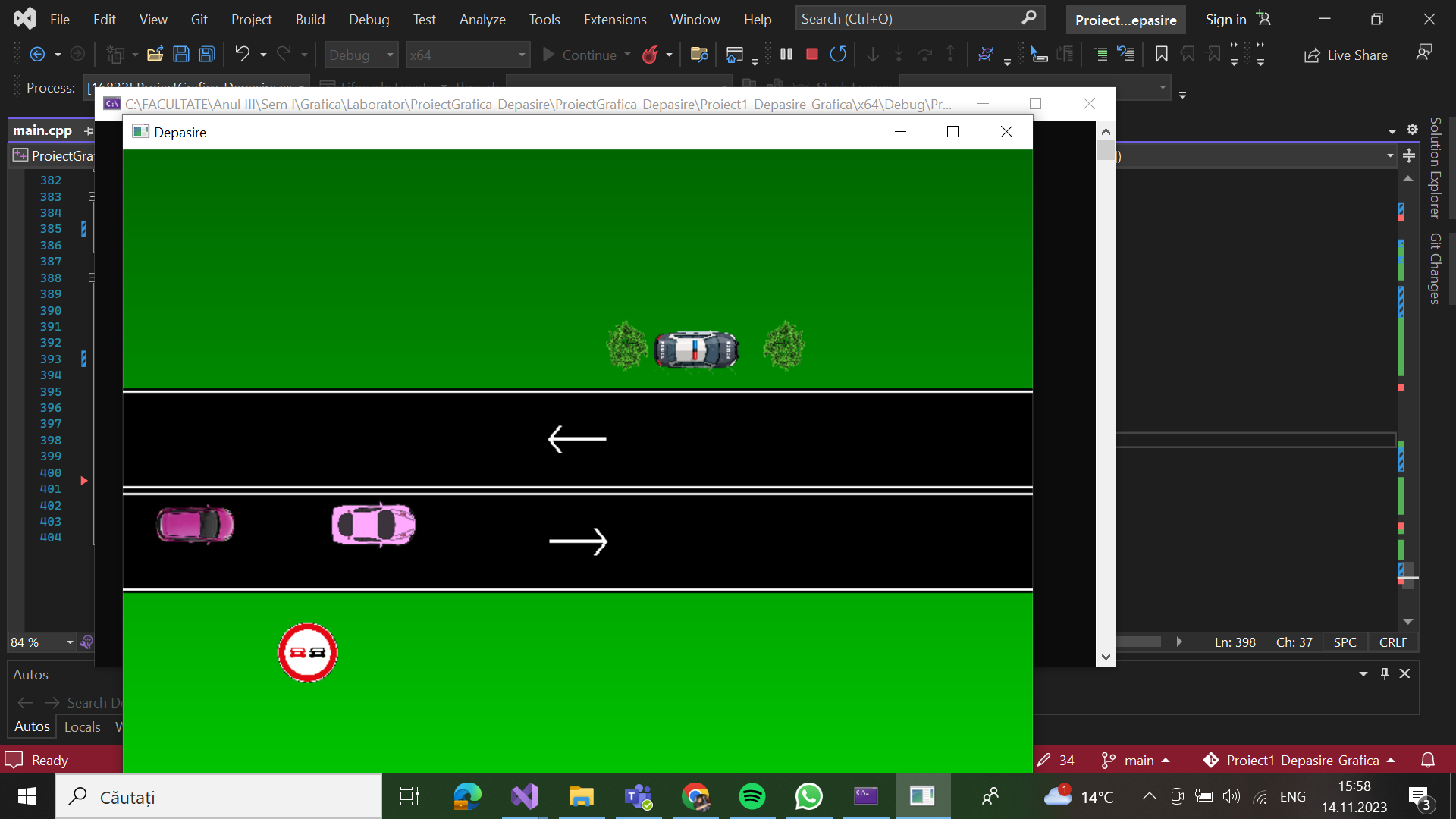
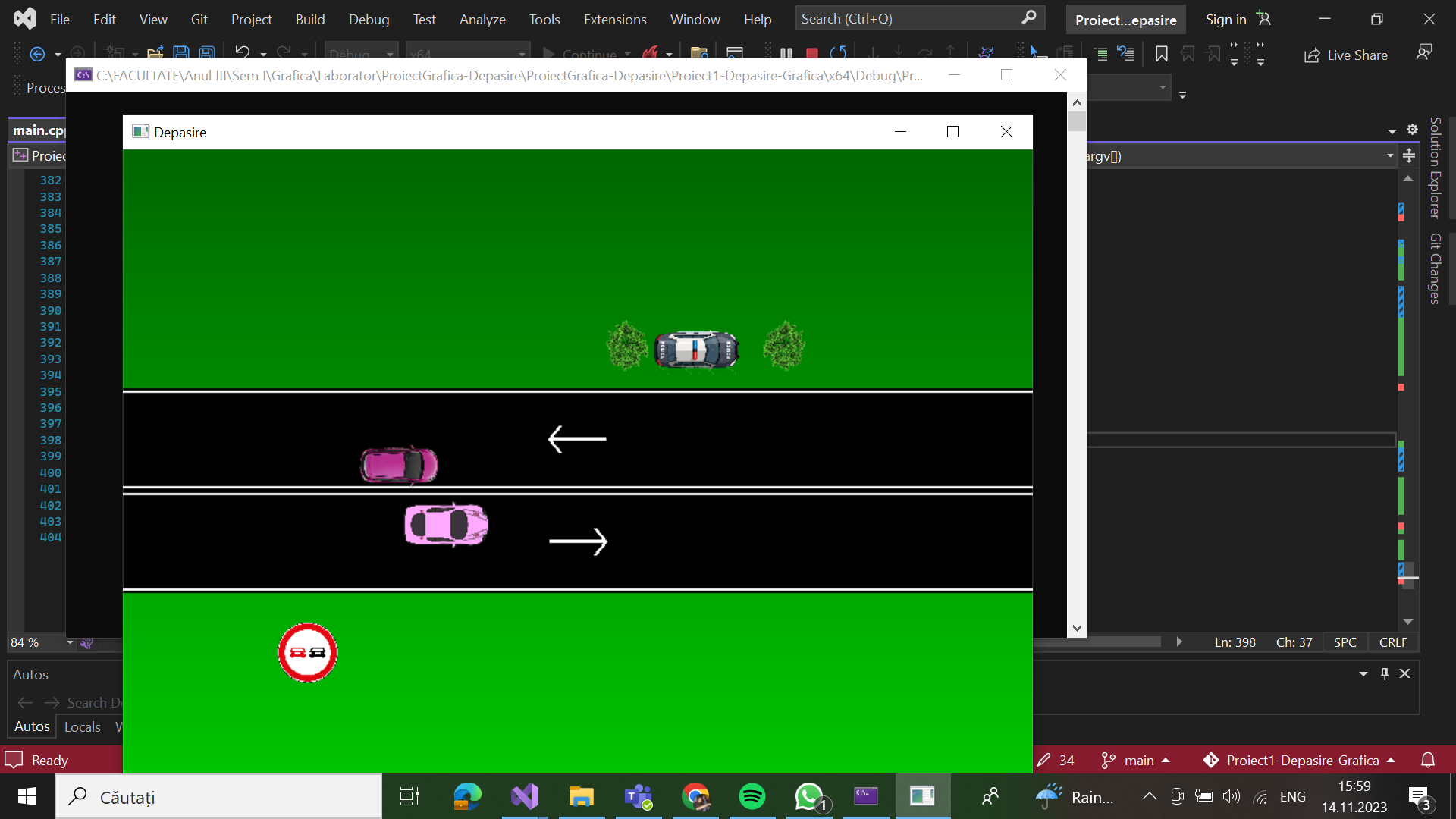
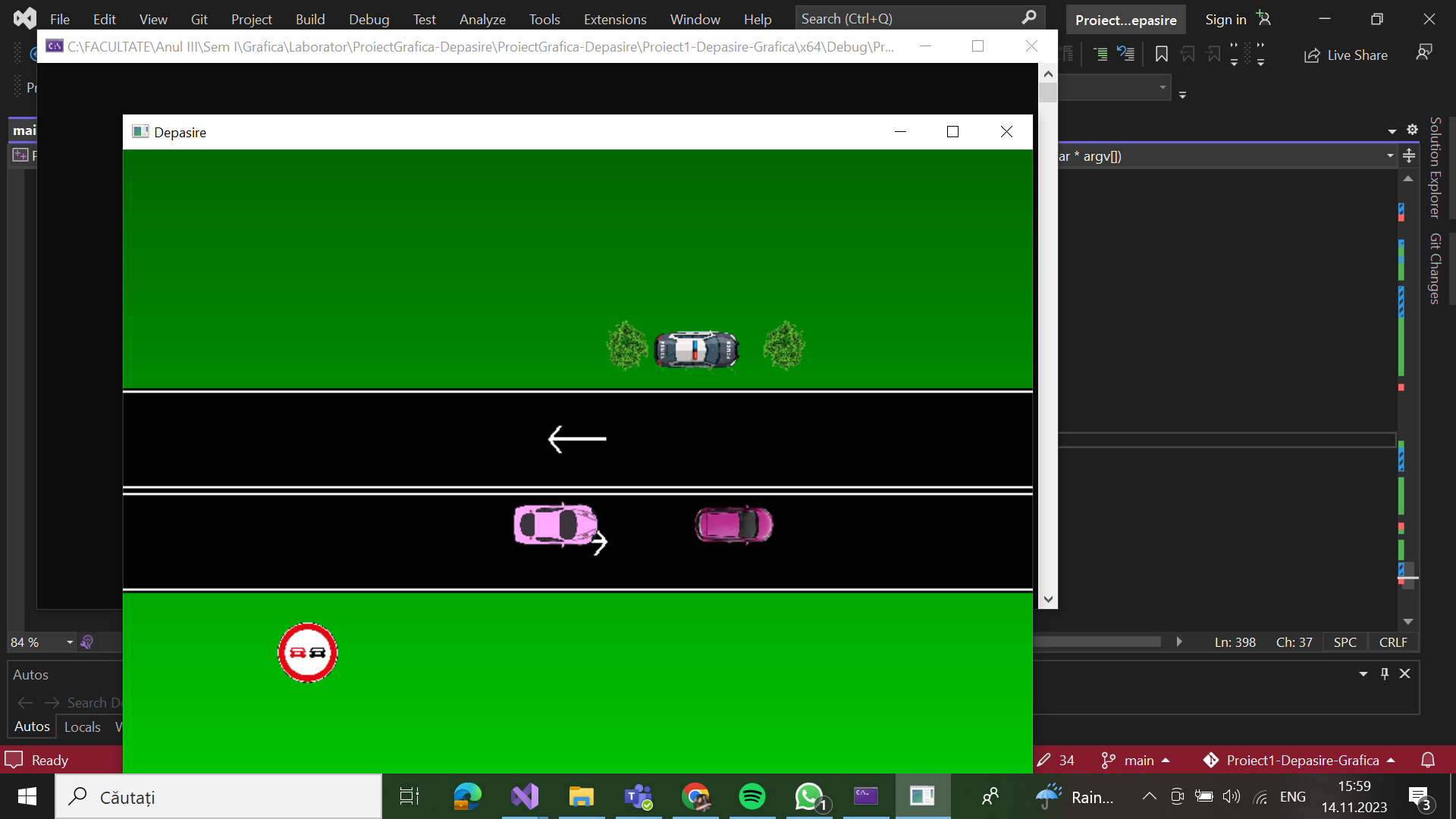
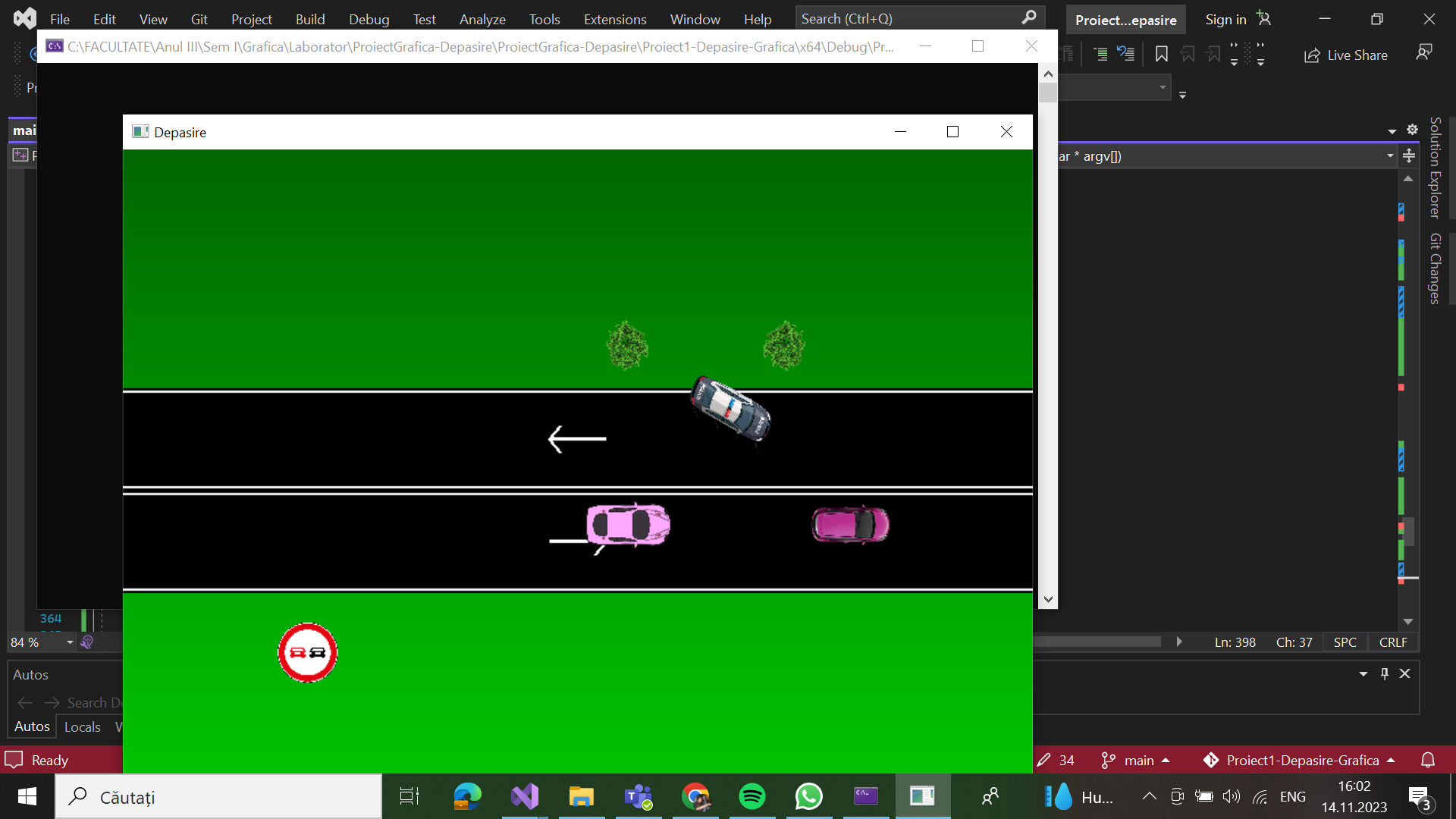
Masina de politie care urmareste masina care depaseste, semnul de circulatie, banda dubla continua, si copacii.

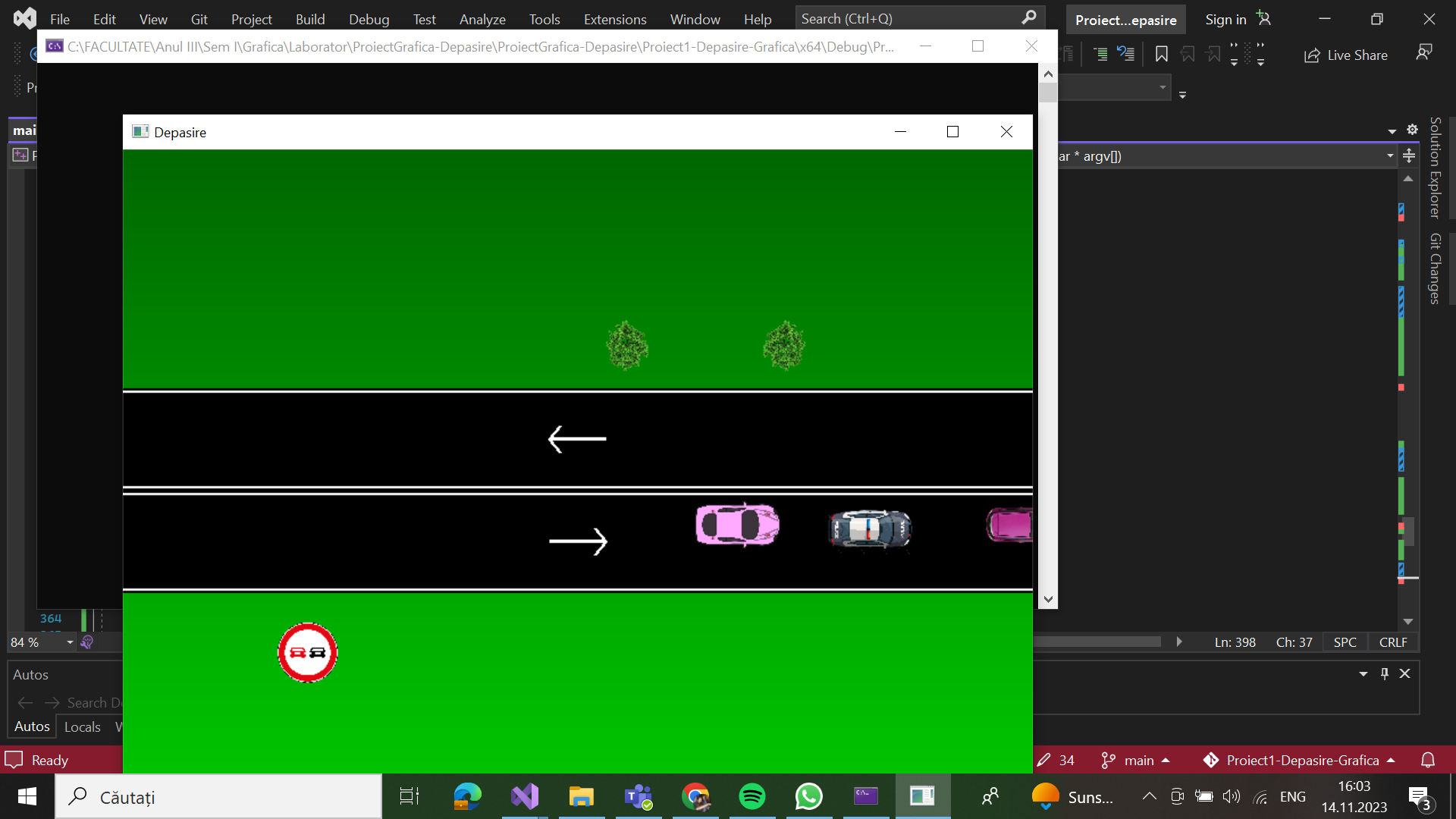
**Echipa si contributii:**

* Dobre Elena Cristina - translatie
* Deleanu Simona Debora - texturare
* Ionescu Andreea Raluca - fundal si documentatie
* Radulescu Alexia Bianca - translatie

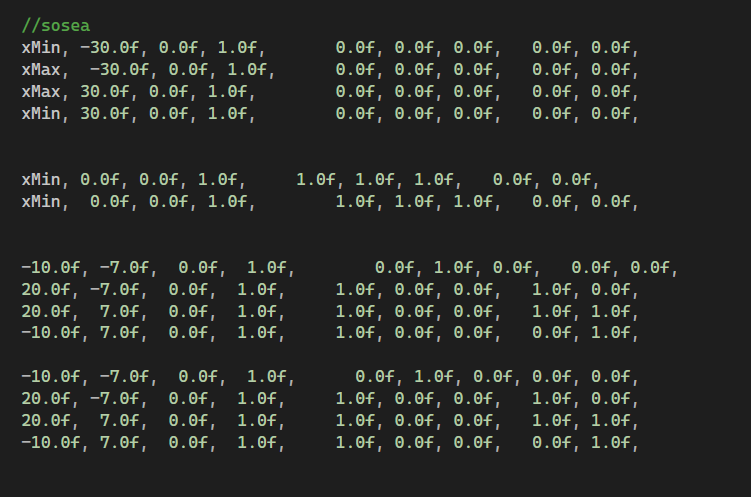
**Conceptul proiectului si originalitate**

Proiectul simuleaza privind de sus o masina care depaseste schimband benzile alta masina, care se deplaseaza uniform. Masina de politie urmareste a doua masina, fiind banda dubla continua. Ne folosim de translatie pentru deplasare si de texturare pentru reprezentarea masinilor. Elementele de originalitate pe care le-am introdus au fost folosirea texturarii pentru masini si politia care urmareste masina pentru ca este semn de circulatie pentru depasire interzisa.

**Miscarea se realizeaza astfel:**



1. Varfurile sunt stocate in vertices prin coordonatele pozitiei, culorii si texturii.



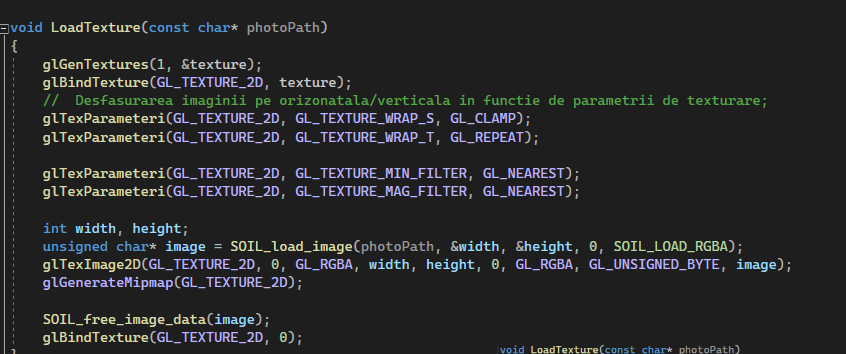
2. Functia LoadTexture incarca imaginile ca texturi pentru masini prin libraria SOIL.

* glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP); -seteaza modul de repetare pe orizontala
* glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);- setează modul de repetare pe verticala
* glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST); - setează filtrarea texturii pentru redimensionarea în mic
* glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST); - setează filtrarea texturii pentru redimensionarea în mare
* int width, height;

unsigned char\* image = SOIL\_load\_image(photoPath, &width, &height, 0,

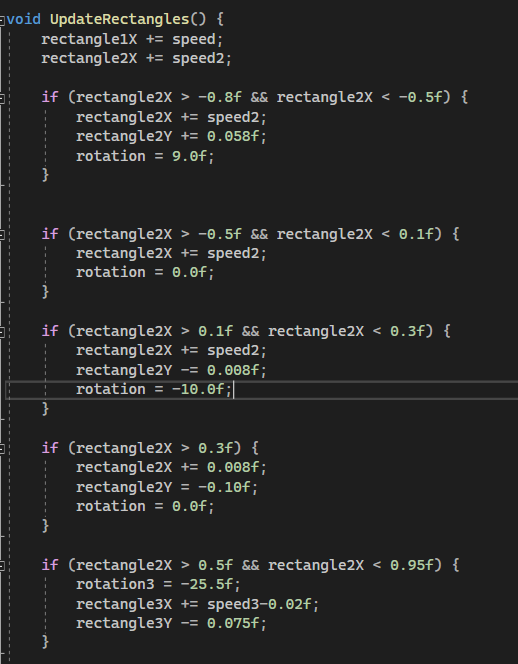
SOIL\_LOAD\_RGBA); - incarca imaginea de la calea specificata (photoPath) si obtine dimensiunile si datele imaginii

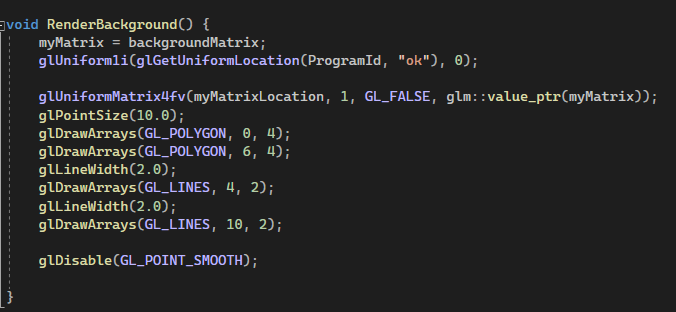
* glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, width, height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, image); - trimite datele imaginii încarcate catre OpenGL pentru a crea textura



3. Functia UpdateRectangles() schimba positia masinilor (“rectangle1X” si “rectangle2X”)

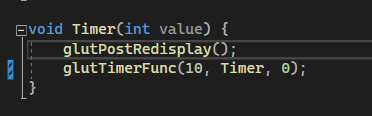
in functie de vitezele “speed” si “speed2”, a doua fiind viteza mai mare a masinii care depaseste. Contine si logica de deplasare a masinii 2 pentru a si schimba banzile.



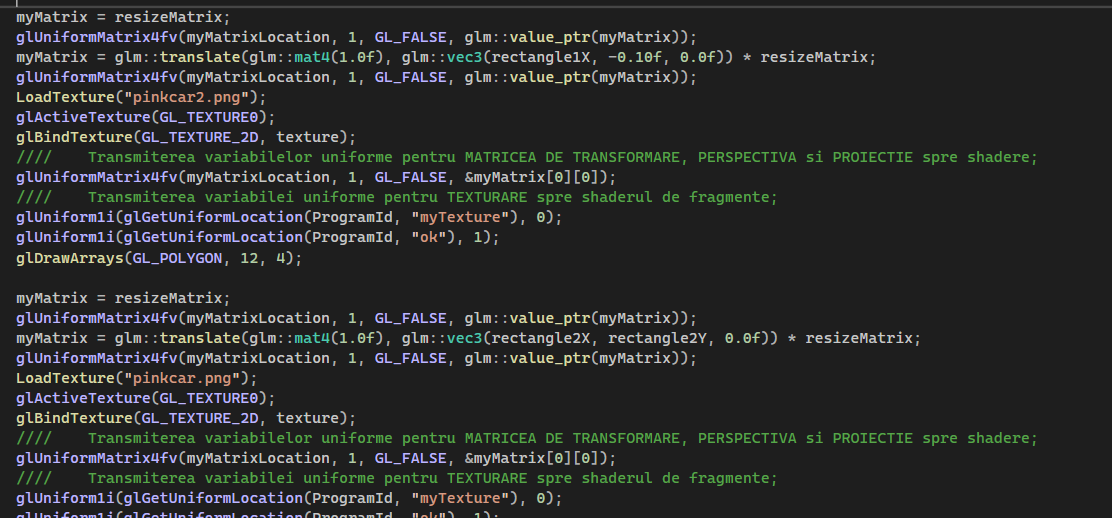
3. Functia RenderBackground() se ocupa de generarea elementelor de pe fundal separat pentru a se aplica texturarea corect (ok este setat la 0)

4. Functia Timer() e folosita pentru a crea imaginea fluida de miscare a masinilor.

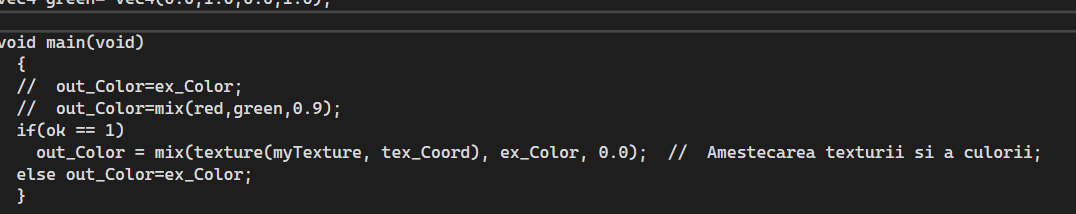
* glutPostRedisplay() - la fiecare interval de timp se redeseneaza scenele si se actualizeaza afisajul.
* glutTimerFunc(10, Timer, 0) - la un interval de 10 de milisecunde, se declanseaza “ glutPostRedisplay()” pentru redesenare.

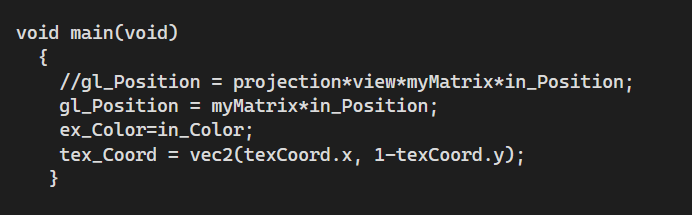


5. In RenderFunction() sunt generate masinile, sunt realizate translatiile si texturarea.

* glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(myMatrix)); - se trimite matricea de transformare catre shader
* myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(rectangle1X, -0.10f, 0.0f)) \* resizeMatrix; - se aplica o translatie la matricea de redimensionare pentru prima masina
* LoadTexture("pinkcar2.png"); -se încarca textura pentru aceasta masina
* glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1); - setarea valorii "ok" la 1, pentru activarea texturarii

6. In interiorul fisierului “03\_05\_Shader.frag” cand valoarea ok este 1 trebuie efectuat blendul cu textura. Este setat al treilea parametru la 0.0 insemnanand ca este full blend.

7. In interiorul fisierului “03\_05\_Shader.vert”, in linia tex\_Coord = vec2(texCoord.x, 1 - texCoord.y); se transforma coordonatele texturii pentru a fi consistente cu conventia OpenGL.



**MAIN**

#include <GL/glew.h>

#include <GL/freeglut.h>

#include <stdio.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"

#include "SOIL.h"

GLuint VaoId, VboId, ProgramId, myMatrixLocation, texture;

glm::mat4 myMatrix, resizeMatrix, backgroundMatrix;

float xMin = -160.0f, xMax = 160.0f, yMin = -100.0f, yMax = 100.0f;

float rectangle1X = -1.0f; // Initial X position of the first rectangle

float rectangle2X = -1.6f; // Initial X position of the second rectangle

float rectangle2Y = -0.10f;

float rectangle3X = 0.23f; //police

float rectangle3Y = 0.4f; //police

float speed = 0.040f; // Speed of movement for 1

float speed2 = 0.056f; // Speed of movement for 1

float speed3 = 0.058f;

float angle = 0; // Unghiul de rotire al patratului;

float tx = 0; float ty = 0; float auxtx; float auxangle;

float rotation = 0.0f;

float rotation3 = 0.0f;

void LoadTexture(const char\* photoPath)

{

glGenTextures(1, &texture);

glBindTexture(GL\_TEXTURE\_2D, texture);

// Desfasurarea imaginii pe orizonatala/verticala in functie de parametrii de texturare;

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

int width, height;

unsigned char\* image = SOIL\_load\_image(photoPath, &width, &height, 0, SOIL\_LOAD\_RGBA);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, width, height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, image);

glGenerateMipmap(GL\_TEXTURE\_2D);

SOIL\_free\_image\_data(image);

glBindTexture(GL\_TEXTURE\_2D, 0);

}

void UpdateRectangles() {

rectangle1X += speed;

rectangle2X += speed2;

if (rectangle2X > -0.8f && rectangle2X < -0.5f) {

rectangle2X += speed2;

rectangle2Y += 0.058f;

rotation = 9.0f;

}

if (rectangle2X > -0.5f && rectangle2X < 0.1f) {

rectangle2X += speed2;

rotation = 0.0f;

}

if (rectangle2X > 0.1f && rectangle2X < 0.3f) {

rectangle2X += speed2;

rectangle2Y -= 0.008f;

rotation = -10.0f;

}

if (rectangle2X > 0.3f) {

rectangle2X += 0.008f;

rectangle2Y = -0.10f;

rotation = 0.0f;

}

if (rectangle2X > 0.5f && rectangle2X < 0.95f) {

rotation3 = -25.5f;

rectangle3X += speed3-0.02f;

rectangle3Y -= 0.075f;

}

if (rectangle3X > 0.48f) {

rotation3 = 0.0f;

rectangle3X += 0.058f;

}

}

void CreateShaders() {

ProgramId = LoadShaders("03\_05\_Shader.vert", "03\_05\_Shader.frag");

glUseProgram(ProgramId);

}

void CreateVBO(void)

{

// Coordonatele varfurilor;

static const GLfloat Vertices[] =

{

// Cele 4 varfuri din colturi;

xMin, yMin, 0.0f, 1.0f, 0.0f, 0.8f, 0.0f, 0.0f, 0.0f,

xMax, yMin, 0.0f, 1.0f, 0.0f, 0.8f, 0.0f, 0.0f, 0.0f,

xMax, yMax, 0.0f, 1.0f, 0.0f, 0.4f, 0.0f, 0.0f, 0.0f,

xMin, yMax, 0.0f, 1.0f, 0.0f, 0.4f, 0.0f, 0.0f, 0.0f,

// Varfuri pentru axe;

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

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

//sosea

xMin, -30.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,

xMax, -30.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,

xMax, 30.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,

xMin, 30.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,

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

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

-10.0f, -7.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,

20.0f, -7.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,

20.0f, 7.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,

-10.0f, 7.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

-10.0f, -7.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,

20.0f, -7.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,

20.0f, 7.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,

-10.0f, 7.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

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

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

xMin, -1.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

xMax, -1.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

xMin, -29.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

xMax, -29.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

xMin, 29.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

xMax, 29.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

-10.0f, 15.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

10.0f, 15.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

-10.0f, -15.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

10.0f, -15.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

10.0f, -15.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

6.0f, -11.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

10.0f, -15.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

6.0f, -19.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

-10.0f, 15.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

-6.0f, 11.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

-10.0f, 15.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

-6.0f, 19.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

65.0f , 35.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

80.0f, 35.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f,

80.0f, 50.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f,

65.0f, 50.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f,

25.0f , 35.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,

10.0f, 35.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f,

10.0f, 50.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f,

25.0f, 50.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f,

//semn circulatie

-80, -35, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,

-110, -35, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f,

-110, -60, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f,

-80, -60, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,

//masina politie

-10.0f, -7.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,

20.0f, -7.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,

20.0f, 7.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,

-10.0f, 7.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

};

// Transmiterea datelor prin buffere;

glGenVertexArrays(1, &VaoId);

glBindVertexArray(VaoId);

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);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 4, GL\_FLOAT, GL\_FALSE, 9 \* sizeof(GLfloat), (GLvoid\*)0);

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

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 9 \* sizeof(GLfloat), (GLvoid\*)(4 \* sizeof(GLfloat)));

// Se asociaza atributul (2 = texturare) pentru shader;

glEnableVertexAttribArray(2);

glVertexAttribPointer(2, 2, GL\_FLOAT, GL\_FALSE, 9 \* sizeof(GLfloat), (GLvoid\*)(7 \* sizeof(GLfloat)));

}

void DestroyShaders(void)

{

glDeleteProgram(ProgramId);

}

void DestroyVBO(void)

{

glDisableVertexAttribArray(1);

glDisableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glDeleteBuffers(1, &VboId);

glBindVertexArray(0);

glDeleteVertexArrays(1, &VaoId);

}

void Cleanup() {

DestroyShaders();

DestroyVBO();

}

void Initialize() {

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // 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;

myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");

glUniform1i(glGetUniformLocation(ProgramId, "ok"), 0);

// Dreptunghiul "decupat";

resizeMatrix = glm::ortho(xMin, xMax, yMin, yMax);

backgroundMatrix = glm::ortho(xMin, xMax, yMin, yMax); // Matrice pentru fundal și linia albă

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

}

void RenderBackground() {

myMatrix = backgroundMatrix;

glUniform1i(glGetUniformLocation(ProgramId, "ok"), 0);

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(myMatrix));

glPointSize(10.0);

glDrawArrays(GL\_POLYGON, 0, 4);

glDrawArrays(GL\_POLYGON, 6, 4);

//glLineWidth(2.0);

//glDrawArrays(GL\_LINES, 4, 2);

glLineWidth(2.0);

glDrawArrays(GL\_LINES, 10, 2);

glLineWidth(2.0);

glDrawArrays(GL\_LINES, 20, 2);

glLineWidth(2.0);

glDrawArrays(GL\_LINES, 22, 2);

glLineWidth(2.0);

glDrawArrays(GL\_LINES, 24, 2);

glLineWidth(2.0);

glDrawArrays(GL\_LINES, 26, 2);

glLineWidth(3.0);

glDrawArrays(GL\_LINES, 28, 2);

glLineWidth(3.0);

glDrawArrays(GL\_LINES, 30, 2);

glLineWidth(3.0);

glDrawArrays(GL\_LINES, 32, 2);

glLineWidth(3.0);

glDrawArrays(GL\_LINES, 34, 2);

glLineWidth(3.0);

glDrawArrays(GL\_LINES, 36, 2);

glLineWidth(3.0);

glDrawArrays(GL\_LINES, 38, 2);

glDisable(GL\_POINT\_SMOOTH);

}

void RenderFunction() {

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

RenderBackground(); // Desenează fundalul

UpdateRectangles();

LoadTexture("semncirc.png");

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, texture);

//// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;

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

//// Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de fragmente;

glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);

glDrawArrays(GL\_POLYGON, 48, 4);

LoadTexture("copacdesus.png");

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, texture);

//// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;

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

//// Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de fragmente;

glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);

glDrawArrays(GL\_POLYGON, 40, 4);

LoadTexture("copacdesus.png");

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, texture);

//// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;

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

//// Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de fragmente;

glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);

glDrawArrays(GL\_POLYGON, 44, 4);

myMatrix = resizeMatrix;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(myMatrix));

myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(rectangle1X, -0.10f, 0.0f)) \* resizeMatrix;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(myMatrix));

LoadTexture("pinkcar2.png");

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, texture);

//// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;

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

//// Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de fragmente;

glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);

glDrawArrays(GL\_POLYGON, 12, 4);

myMatrix = resizeMatrix;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(myMatrix));

myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(rectangle2X, rectangle2Y, 0.0f)) \* resizeMatrix;

myMatrix = glm::rotate(myMatrix, glm::radians(rotation), glm::vec3(0.0f, 0.0f, 1.0f));

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(myMatrix));

LoadTexture("pinkcar.png");

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, texture);

//// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;

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

//// Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de fragmente;

glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);

glDrawArrays(GL\_POLYGON, 16, 4);

//masina politie

myMatrix = resizeMatrix;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(myMatrix));

myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(rectangle3X, rectangle3Y, 0.0f)) \* resizeMatrix;

myMatrix = glm::rotate(myMatrix, glm::radians(rotation3), glm::vec3(0.0f, 0.0f, 1.0f));

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(myMatrix));

LoadTexture("masinapolitie.png");

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, texture);

//// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;

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

//// Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de fragmente;

glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);

glDrawArrays(GL\_POLYGON, 52, 4);

glutSwapBuffers(); // Inlocuieste imaginea deseneata in fereastra cu cea randata;

glFlush();

}

void Timer(int value) {

glutPostRedisplay();

glutTimerFunc(10, Timer, 0);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 600);

glutInitWindowPosition(100, 100);

glutCreateWindow("Depasire");

glewInit();

Initialize();

glutDisplayFunc(RenderFunction);

glutCloseFunc(Cleanup);

glutTimerFunc(0, Timer, 0);

glutMainLoop();

return 0;

}

**“03\_05\_Shader.vert”**

//

// ================================================

// | Grafica pe calculator |

// ================================================

// | Laboratorul III - 03\_05\_Shader.vert |

// =======================================

//

// Shaderul de varfuri / Vertex shader - afecteaza geometria scenei;

//

#version 330 // Versiunea GLSL;

// Variabile de intrare (dinspre programul principal);

layout (location = 0) in vec4 in\_Position; // Se preia din buffer de pe prima pozitie (0) atributul care contine coordonatele;

layout (location = 1) in vec4 in\_Color; // Se preia din buffer de pe a doua pozitie (1) atributul care contine culoarea;

layout (location=2) in vec2 texCoord; // Se preia din buffer de pe a treia pozitie (2) atributul care contine textura;

// Variabile de iesire;

out vec4 gl\_Position; // Transmite pozitia actualizata spre programul principal;

out vec4 ex\_Color; // Transmite culoarea (de modificat in Shader.frag);

out vec2 tex\_Coord; // Transmite textura (de modificat in Shader.frag);

// Variabile uniforme;

uniform mat4 myMatrix;

uniform mat4 view;

uniform mat4 projection;

void main(void)

{

//gl\_Position = projection\*view\*myMatrix\*in\_Position;

gl\_Position = myMatrix\*in\_Position;

ex\_Color=in\_Color;

tex\_Coord = vec2(texCoord.x, 1-texCoord.y);

}

**“03\_05\_Shader.frag”**

//

// ================================================

// | Grafica pe calculator |

// ================================================

// | Laboratorul III - 03\_05\_Shader.frag |

// =======================================

//

// Shaderul de fragment / Fragment shader - afecteaza culoarea pixelilor;

//

#version 330 // Versiunea GLSL;

// Variabile de intrare (dinspre Shader.vert);

in vec4 ex\_Color;

in vec2 tex\_Coord; // Coordonata de texturare;

// Variabile de iesire (spre programul principal);

out vec4 out\_Color; // Culoarea actualizata;

// Variabile uniforme;

uniform sampler2D myTexture;

uniform int ok;

// Variabile pentru culori;

vec4 red = vec4(1.0,0.0,0.0,1.0);

vec4 green= vec4(0.0,1.0,0.0,1.0);

void main(void)

{

// out\_Color=ex\_Color;

// out\_Color=mix(red,green,0.9);

if(ok == 1)

out\_Color = mix(texture(myTexture, tex\_Coord), ex\_Color, 0.0); //Amestecarea texturii si a culorii;

else out\_Color=ex\_Color;

}