Proiect 1 - DEPĂȘIRE

Cuprins

Cerință

- Simulați o "depășire": o masină / un dreptunghi se deplasează uniform (prin translație),
  un alt dreptunghi vine din spate (tot prin translații/rotații), la un moment dat intr în depășire, apoi trece în fața primului.

Abordare

- Acest proiect vine ca o transpunere a elementelor de grafică 2D învățate într-un joc ce presupune depășirea mașinilor de pe sensul de mers.
- “Jucătorul” va putea controla direcția unei mașini prin intermediul tastelor săgeți stânga/dreapta, dar și viteza acesteia prin intermediul tastei săgeată sus și astfel va

putea depăși mașinile care apar pe carosabil pe sensul lui de mers. Coliziunea cu alte mașini va duce la afișarea mesajului “Game Over”.

- Pentru a simula scena propriu-zisă de depășire, asupra mașinii “jucătorului” (menționată în continuare ca “mașina roșie”) vor fi aplicate doar translații stânga/dreapta (translații pe axa Ox). Atât background-ul, cât și celelalte mașini vor fi translatate pe axa Oy, astfel va fi simulată înaintarea mașinii roșii, creându-se o imagine dinamic.

Background

Pentru ideea de fundal infinit, am utilizat două imagini, având aceeași textură. Cea de-a doua o succede pe prima și are, deci, coordonatele colțurilor de jos corespunzătoare celor două de sus ale primei:

```
        -390.0f, -190.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0
         490.0f, -190.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0
         490.0f, +390.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 1
        -390.0f, +390.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1

        -390.0f, +390.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0
         490.0f, +390.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0
         490.0f, 970.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0
        -390.0f,970.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f,0.0f, 1.0f,
```

Pentru desenarea obiectelor, am definit o funcție `BuildObject` având utilitate în refolosirea codului.

```
void BuildObject(glm::vec3 transl, glm::mat4 matrTransl1, const
float startPoint) {

    myMatrix = resizeMatrix;
    matrTransl = glm::translate(glm::mat4(1.0f), transl);
    myMatrix = resizeMatrix * matrTransl * matrTransl1;

    LoadTexture(pathTexture);
    glActiveTexture(GL_TEXTURE0);
```

```
    glBindTexture(GL_TEXTURE_2D, texture);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    //  Transmiterea variabilei uniforme pentru TEXTURARE spre s
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0)
    glDrawArrays(GL_POLYGON, startPoint, 4);
}
```

Pentru background-ul nostru, în `RenderFunction`, apelăm această funcție în modul următor:

```
    BuildObject(translCoordBackground, matrTranslBackground, "ba
    BuildObject(translCoordBackground, matrTranslBackground, "ba
```

Continuitatea fundalului

Pentru a simula o scenă dinamică, am realizat translatarea background-ului, resetându-se valorile la 0 în momentul în care translația este mai mare decât lungimea background-ului folosit.

```
// în funcția updateTranslationCoordonates

translCoordBackground.y -= 45.0f;
if (std::abs(translCoordBackground.y) >= 390.0f - (-190.0f)) {
    translCoordBackground.y = 0.0f;
}
```

Mașini

Definim coordonatele mașinilor:

🚗 Mașina roșie este cea a “jucătorului”.

```
    //Masina verde 1
        105.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
```

```
        115.0f,0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
        115.0f,20.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
        105.0f,20.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,

        //Masina roșie
        105.0f, -120.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f,  0.0f, 0
        115.0f,-120.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0
        115.0f,-100.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0
        105.0f,-100.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0

        //Mașina galbena
        70.0f, +390.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0
        80.0f, +390.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0
        80.0f, 370.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
        70.0f, 370.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f,0.0f, 1.0f,

        //Mașina verde 2
        105.0f, 450.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f,   0.0f, 0
        115.0f, 450.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0
        115.0f, 430.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0
        105.0f, 430.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0
```

Pentru desenarea acestora pe ecran, vom apela funcția `BuildObject` prezentată anterior:

```
// în RenderFunction
  //Masina verde 1
    BuildObject(translCoordCarGreen1, matrTranslGreen1Car, "gree

    //Mașina rosie
    BuildObject(translCoordCarRed, matrTranslRedCar, "redCar.png

    //Mașina galbena
    BuildObject(translCoordCarYellow, matrTranslYellowCar, "yell
```

```
    //Mașina verde 2
    BuildObject(translCoordCarGreen2, matrTranslGreen2Car, "gree
```

Generarea mașinilor pe benzi

Pentru a asigura dinamismul scenei și posibilitatea continuării depășirilor pe o perioadă nedeterminată, atunci când o mașină iese din scena (coordonatele acesteia depășesc coordonatele background-ului), coordonata y de translație va fi reinițializată la 0, iar mașina va reîncepe deplasarea din punctul inițial în care a fost declarată si anume din partea superioară a background-ului. Coordonata x se va stabili aleator (poziționând astfel mașina pe una din cele 2 benzi de pe sens).

```
// în updateTranslationCoordonates()
if (std::abs(translCoordCarYellow.y) >= 390.0f - (-190.0f)) {
            int randomNumber = std::rand() % 2;
            translCoordCarYellow.x = 0.0f;

            if (randomNumber == 0) {
                translCoordCarYellow.x = 0.0f;
            }
            else {
                translCoordCarYellow.x += 35.0f;
            }
            translCoordCarYellow.y = 0.0f;
        }
```

Depășirea

Mașina roșie (mașina “jucătorului”) se poate deplasa stânga/dreapta.

```
// în funcția updateTranslationCoordonates()
if (moveLeft) {
        translCoordCarRed.x -= 5.0f; //coordonata x scade pentru
    }
```

```
    if (moveRight) {
        translCoordCarRed.x += 5.0f; //coordonata x creste pentr
    }
```

Celelalte mașini se vor deplasa în josul fundalului, astfel coordonata y va scădea pentru translația în jos pe axa Oy.

```
    translCoordCarGreen1.y -= 15.0f;
        translCoordCarYellow.y -= 15.0f;
        translCoordCarGreen2.y -= 15.0f;
```

KeyPressed & KeyReleased

Mașina roșie se poate deplasa stânga/dreapta. Celelalte mașini și implicit fundalul se vor deplasa în jos pe axa Oy pentru a reda efectul de înaintare a mașinii roșii.

Aceste translații corespunzătoare mișcărilor descrise se realizează cu ajutorul săgeților. Funcțiile `KeyPressed` și `KeyReleased` gestionează evenimente de apăsare a tastelor săgeată stânga, săgeată dreapta și săgeată sus, ajustând stările variabilelor corespunzătoare și redesenând scena pentru a reflecta schimbările.

```
void KeysPressed(int key, int param2, int param3)
{
    if (key == GLUT_KEY_LEFT) {
        moveLeft = true;
        moveRight = false;
    }

    if (key == GLUT_KEY_RIGHT) {
        moveLeft = false;
        moveRight = true;
    }

    if (key == GLUT_KEY_UP) {
        moveForward = true;
    }
```

```
    glutPostRedisplay();
}

void KeysReleased(int key, int param2, int param3)
{
    if (key == GLUT_KEY_LEFT) {
        moveLeft = false;
        moveRight = false;
    }

    if (key == GLUT_KEY_RIGHT) {
        moveLeft = false;
        moveRight = false;
    }

    if (key == GLUT_KEY_UP) {
        moveForward = false;
    }
}
```

Funcțiile `KeyPressed` și `KeyReleased` vor fi apelate în main:

```
  // în main
  glutSpecialFunc(KeysPressed);       //  Callback cand tasta e
    glutSpecialUpFunc(KeysReleased);    //  Callback cand tasta
```

Coliziunea

Pentru a detecta coliziunea dintre 2 dreptunghiuri (în cazul nostru, mașini), vom utiliza o funcție care primește ca argumente coordonatele mașinilor.

Vom verifica dacă un dreptunghi este pe partea stângă/dreaptă a celuilalt, respectiv deasupra/dedesubtul celuilalt, caz în care obiectele nu se suprapun.

```
bool doOverlap(glm::vec3 translCoordCar2, glm::vec3 translCoordC
{
    if (105.0f + translCoordCar2.x > 80.0f + translCoordCar3.x
        70 + translCoordCar3.x > 115.0f + translCoordCar2.x)
        return false;

    if (-120.0f > 390.0f + translCoordCar3.y ||
        370.0f + translCoordCar3.y > -100.0f)
        return false;

    return true;
}
```

Aceste funcții vor fi apelate în `updateTranslationCoordonates()` și intersecția mașinilor va fi semnalizată cu ajutorul variabilei bool `GameOver.`

```
// în updateTranslationCoordonates
if (doOverlap(translCoordCarRed, translCoordCarYellow) == true
    || doOverlapCar4(translCoordCarRed, translCoordCarGreen2) ==
        || doOverlap(translCoordCarRed, translCoordCarRed) == tr
    || doOverlapCar4(translCoordCarGreen2, translCoordCarRed) ==
        || doOverlapCar4(translCoordCarGreen1, translCoordCarRe
    || doOverlapCar4(translCoordCarRed, translCoordCarGreen1) ==
        //std::cout << "Intersection\n";
        GameOver = true;
```

În cazul în care variabila `GameOver` devine true, astfel a fost detectată o coliziune între mașini, vor apărea imagini sugestive cu scrisul “Game Over” și respectiv cu simularea unei explozii.

```
// în RenderFunction
//EndGame
    if (GameOver == true) {
        BuildObject(translCoordGameOver, matrTranslGameOver, "Ga
        BuildObject(translCoordCollision, matrTranslCollision, "
```

```
    }
```

Coordonatele acestor obiecte:

```
    //GameOver
        -340.0f, 175.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.(
        430.0f, 175.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0
        430.0f, 370.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0
        -340.0f, 370.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.(

        //Collision
        105.0f + translCoordCarRed.x,-120.0f, 0.0f, 1.0f, 1.0f,
        125.0f + translCoordCarRed.x,-120.0f, 0.0f, 1.0f, 1.0f,
        125.0f + translCoordCarRed.x,-100.0f, 0.0f, 1.0f, 1.0f,
        105.0f + translCoordCarRed.x,-100.0f, 0.0f, 1.0f, 1.0f,
```

Modificări ulterioare

Am adăugat mișcarea de rotație pentru mașina roșie în momentul depășirii.

Pentru acest lucru, am folosit o matrice de rotație `matrRot` și am inițializat în funcția KeyPressed unghiul (`angle`) cu valoarea `PI/30` (atunci când este apăsată tasta săgeată stânga) și `-PI/30` (când este apăsată tasta săgeată dreapta).

```
void BuildObject(glm::vec3 transl, glm::mat4 matrTransl1, const char* pathTexture, float startPoint, bool isRedCar) {
    myMatrix = resizeMatrix;
    matrTransl = glm::translate(glm::mat4(1.0f), transl);

    if (isRedCar == false) {
        myMatrix = resizeMatrix * matrTransl * matrTransl1;

    }

    else {
        matrRot = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0f, 0.0f, 1.0f));
        myMatrix = resizeMatrix * matrRot * matrTransl * matrTransl1;

    }


    LoadTexture(pathTexture);
    glActiveTexture(GL_TEXTURE0);
```

Referințe

https://www.geeksforgeeks.org/find-two-rectangles-overlap/

Componența echipei

- Băicoianu Bianca, grupa 351
- Buruiană Cosmina, grupa 332
- Georgescu Miruna, grupa 332
- Neaga Maria, grupa 332

Gestionarea sarcinilor

- fundal “infinit” prin translație - Miruna
- generarea mașinilor & poziționare aleator pe una din benzile de pe sens - Bianca
- controlul mașinii din taste (KeyPressed & KeyReleased) - Maria
- realizarea coliziunii - Cosmina
- folosirea texturilor - toate
- documentație - toate

Cod sursa

Fișierul .cpp

```
//
// ================================================
// | Grafica pe calculator                        |
// ================================================
//  Biblioteci

#include <windows.h>        //  Utilizarea functiilor de sistem
#include <stdlib.h>         //  Biblioteci necesare pentru citir
#include <stdio.h>
#include <GL/glew.h>        //  Defineste prototipurile functii
#include <GL/freeglut.h>    //  Include functii pentru:
                            //  - gestionarea ferestrelor si eve
```

```
                            //  - desenarea de primitive grafice
                            //  - crearea de meniuri si submeniu
#include "loadShaders.h"    //  Fisierul care face legatura intr
#include "glm/glm.hpp"      //  Bibloteci utilizate pentru trans
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtx/transform.hpp"
#include "glm/gtc/type_ptr.hpp"
#include <glm/gtx/vector_angle.hpp>
#include "SOIL.h"
#include <iostream>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string>

//  Identificatorii obiectelor de tip OpenGL;
GLuint
VaoId,
VboId,
EboId,
ProgramId,
myMatrixLocation,
viewLocation,
projLocation,
matrRotlLocation,
codColLocation;

GLuint
texture;
//  Dimensiunile ferestrei de afisare;
GLfloat
winWidth = 900, winHeight = 600;
//  Variabile catre matricile de transformare;
glm::mat4
myMatrix, resizeMatrix, matrTransl, matrScale,
matrDeplasare, matrTrans, matrRot, matrTranslBackground, matrTra
```

```
matrTranslGameOver, matrTranslCollision;

//  Variabila ce determina schimbarea culorii pixelilor in shade
int codCol;
float
xMin = -400, xMax = 500, yMin = -200, yMax = 400;
float PI = 3.14;
float i = 0.0, alpha = 0.0, step = 0.3, beta = 0.002;
float angle;

glm::vec3 translCoordCarRed(0.0f, 0.0f, 0.0f); //coordonatele tr
glm::vec3 translCoordCarGreen1(0.0f, 0.0f, 0.0f);
glm::vec3 translCoordCarYellow(0.0f, 0.0f, 0.0f);
glm::vec3 translCoordCarGreen2(0.0f, 0.0f, 0.0f);
glm::vec3 translCoordBackground(0.0f, 0.0f, 0.0f);
glm::vec3 translCoordGameOver(0.0f, 0.0f, 0.0f);
glm::vec3 translCoordCollision(0.0f, 0.0f, 0.0f);


//  Crearea si compilarea obiectelor de tip shader;
//  Trebuie sa fie in acelasi director cu proiectul actual;
//  Shaderul de varfuri / Vertex shader - afecteaza geometria sc
//  Shaderul de fragment / Fragment shader - afecteaza culoarea

bool moveLeft{ false }, moveRight{ false }, moveForward{ false }


bool doOverlap(glm::vec3 translCoordCar2, glm::vec3 translCoordC
{

    // If one rectangle is on left side of other
    if (105.0f + translCoordCar2.x > 80.0f + translCoordCar3.x
        70 + translCoordCar3.x > 115.0f + translCoordCar2.x)
        return false;

    //If one rectangle is above other
```

```
    if (-120.0f > 390.0f + translCoordCar3.y
        || 370.0f + translCoordCar3.y > -100.0f)
        return false;

    return true;
}

bool doOverlapCar4(glm::vec3 translCoordCar2, glm::vec3 translC
{

    // If one rectangle is on left side of other
    if (105.0f + translCoordCar2.x > 115.0f + translCoordCar4.x
        105 + translCoordCar4.x > 115.0f + translCoordCar2.x)
        return false;

    //If one rectangle is above other
    if (-120.0f > 450.0f + translCoordCar4.y
        || 430.0f + translCoordCar4.y > -100.0f)
        return false;

    return true;
}


bool GameOver = false;
void updateTranslationCoordonates()
{
    std::srand(std::time(0));

    // Generare număr aleatoriu în intervalul [1, 100]

    if (moveLeft) {
        translCoordCarRed.x -= 5.0f; //coordonata x scade pentru
    }

    if (moveRight) {
```

```
        translCoordCarRed.x += 5.0f; //coordonata x creste pentr
    }

    if (moveForward) {
        translCoordCarGreen1.y -= 15.0f;
        translCoordCarYellow.y -= 15.0f;
        translCoordCarGreen2.y -= 15.0f;

        translCoordBackground.y -= 45.0f;
        if (std::abs(translCoordBackground.y) >= 390.0f - (-190
            translCoordBackground.y = 0.0f;
        }

        if (std::abs(translCoordCarYellow.y) >= 390.0f - (-190.0
            int randomNumber = std::rand() % 2;
            translCoordCarYellow.x = 0.0f;

            if (randomNumber == 0) {
                translCoordCarYellow.x = 0.0f;
            }
            else {
                translCoordCarYellow.x += 35.0f;
            }
            translCoordCarYellow.y = 0.0f;
        }
        if (std::abs(translCoordCarGreen2.y) >= 390.0f - (-190.0
            int randomNumber = std::rand() % 2;
            translCoordCarGreen2.x = 0.0f;

            if (randomNumber == 0) {
                translCoordCarGreen2.x = 0.0f;
            }
            else {
                translCoordCarGreen2.x -= 35.0f;
            }
            translCoordCarGreen2.y = 0.0f;
```

```
        }

        if (std::abs(translCoordCarGreen1.y) >= 390.0f - (-190.0
            int randomNumber = std::rand() % 2;
            translCoordCarGreen1.x = 0.0f;

            if (randomNumber == 0) {
                translCoordCarGreen1.x = 0.0f;
            }
            else {
                translCoordCarGreen1.x -= 35.0f;
            }
            translCoordCarGreen1.y = 490.0f;
        }
    }
    translCoordCarGreen1.y -= 430.0f;
    if (doOverlap(translCoordCarRed, translCoordCarYellow) == tr
        || doOverlap(translCoordCarRed, translCoordCarRed) == tr
        || doOverlapCar4(translCoordCarGreen1, translCoordCarRed
        //std::cout << "Intersection\n";
        GameOver = true;

    }

    translCoordCarGreen1.y += 430.0f;
}


void KeysPressed(int key, int param2, int param3)
{
    if (key == GLUT_KEY_LEFT) {
        moveLeft = true;
        moveRight = false;
    }

    if (key == GLUT_KEY_RIGHT) {
```

```
        moveLeft = false;
        moveRight = true;
    }

    if (key == GLUT_KEY_UP) {
        moveForward = true;
    }

    glutPostRedisplay();
}

void KeysReleased(int key, int param2, int param3)
{
    if (key == GLUT_KEY_LEFT) {
        moveLeft = false;
        moveRight = false;
        angle = PI / 30;
    }

    if (key == GLUT_KEY_RIGHT) {
        moveLeft = false;
        moveRight = false;
        angle = - PI / 30;
    }

    if (key == GLUT_KEY_UP) {
        moveForward = false;
    }
}

void CreateShaders(void)
{
    ProgramId = LoadShaders("03_03_Shader.vert", "03_03_Shader.f
    glUseProgram(ProgramId);

}
```

```
void LoadTexture(const char* photoPath)
{
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    //  Desfasurarea imaginii pe orizonatala/verticala in functi
    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_NEA
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEA

    int width, height;
    unsigned char* image = SOIL_load_image(photoPath, &width, &h
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_
    glGenerateMipmap(GL_TEXTURE_2D);

    SOIL_free_image_data(image);
    glBindTexture(GL_TEXTURE_2D, 0);

}

//  Se initializeaza un Vertex Buffer Object (VBO) pentru tranfe
//  In acesta se stocheaza date despre varfuri (coordonate, culo
void CreateVBO(void)
{

    //  Coordonatele varfurilor;
    GLfloat Vertices[] = {

        -390.0f, -190.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0
         490.0f, -190.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0
         490.0f, +390.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 1
        -390.0f, +390.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1
```

```
        -390.0f, +390.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0
         490.0f, +390.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0
         490.0f, 970.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0
        -390.0f,970.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f,0.0f, 1.0f,

        //Masina verde
        105.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
        115.0f,0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,//
        115.0f,20.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,/
        105.0f,20.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,/

        //Masina roșie
        105.0f, -120.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f,  0.0f, 0
        115.0f,-120.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f
        115.0f,-100.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f
        105.0f,-100.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f

        //Mașina galbena
        70.0f, +390.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f
        80.0f, +390.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0f
        80.0f, 370.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
        70.0f, 370.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f,0.0f, 1.0f,

        //Mașina verde
        105.0f, 450.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f,   0.0f, 0
        115.0f, 450.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f
        115.0f, 430.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f
        105.0f, 430.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f

        //GameOver
        -340.0f, 175.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0
        430.0f, 175.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f
        430.0f, 370.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f
        -340.0f, 370.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0

        //collision
```

```
        105.0f + translCoordCarRed.x,-120.0f, 0.0f, 1.0f, 1.0f,
        125.0f + translCoordCarRed.x,-120.0f, 0.0f, 1.0f, 1.0f,
        125.0f + translCoordCarRed.x,-100.0f, 0.0f, 1.0f, 1.0f,
        105.0f + translCoordCarRed.x,-100.0f, 0.0f, 1.0f, 1.0f,

    };

    GLuint Indices[] = {
      0, 1, 2,  //  Primul triunghi;
      3 ,1,//  Al doilea triunghi;
      4,5,6,
      7,5,
      8,9,10,11,
      12,13,14,15,
      16,17,18,19,
      20,21,22,23,
      24,25,26,27,
      28,29,30,31
    };

    //  Transmiterea datelor prin buffere;

    //  Se creeaza / se leaga un VAO (Vertex Array Object) - uti
    glGenVertexArrays(1, &VaoId);
    glBindVertexArray(VaoId);

    //  Se creeaza un buffer pentru VARFURI - COORDONATE, CULORI
    glGenBuffers(1, &VboId);
    glBindBuffer(GL_ARRAY_BUFFER, VboId);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL

    //  Se creeaza un buffer pentru INDICI;
    glGenBuffers(1, &EboId);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indic
```

```
    //  Punctele sunt "copiate" in bufferul curent;
    //  Se asociaza atributul (0 = coordonate) pentru shader;
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 9 * sizeof(G
    //  Se asociaza atributul (1 =  culoare) pentru shader;
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 9 * sizeof(G
    //  Se asociaza atributul (2 =  texturare) pentru shader;
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 9 * sizeof(G
}

//  Elimina obiectele de tip shader dupa rulare;
void DestroyShaders(void)
{
    glDeleteProgram(ProgramId);
}

///  Eliminarea obiectelor de tip VBO dupa rulare;
void DestroyVBO(void)
{
    //  Eliberarea atributelor din shadere (pozitie, culoare, te
    glDisableVertexAttribArray(2);
    glDisableVertexAttribArray(1);
    glDisableVertexAttribArray(0);

    //  Stergerea bufferelor pentru VARFURI (Coordonate, Culori,
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDeleteBuffers(1, &VboId);
    glDeleteBuffers(1, &EboId);

    //  Eliberaea obiectelor de tip VAO;
    glBindVertexArray(0);
    glDeleteVertexArrays(1, &VaoId);
}
```

```
//  Functia de eliberare a resurselor alocate de program;
void Cleanup(void)
{
    DestroyShaders();
    DestroyVBO();
}

//  Setarea parametrilor necesari pentru fereastra de vizualizar
void Initialize(void)
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);       //  Culoarea de
    CreateShaders();                            //  Initilizarea
    //  Instantierea variabilelor uniforme pentru a "comunica" c
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix
    resizeMatrix = glm::ortho(xMin, xMax, yMin, yMax);
    matrTranslRedCar = glm::translate(glm::mat4(1.0f), glm::vec3
    matrTranslBackground = glm::translate(glm::mat4(1.0f), glm:
    matrTranslGreen1Car = glm::translate(glm::mat4(1.0f), glm::v
    matrTranslYellowCar = glm::translate(glm::mat4(1.0f), glm::v
    matrTranslGreen2Car = glm::translate(glm::mat4(1.0f), glm::v
    matrTranslGameOver = glm::translate(glm::mat4(1.0f), glm::ve
    matrTranslCollision = glm::translate(glm::mat4(1.0f), glm::v

    //  Pentru shaderul de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0)
}

void BuildObject(glm::vec3 transl, glm::mat4 matrTransl1, const

    myMatrix = resizeMatrix;
    matrTransl = glm::translate(glm::mat4(1.0f), transl);
    myMatrix = resizeMatrix * matrTransl * matrTransl1;

    if (isRedCar == false) {
        myMatrix = resizeMatrix * matrTransl * matrTransl1;
    }
```

```
    else {
        matrRot = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(
        myMatrix = resizeMatrix * matrRot * matrTransl * matrTra
    }

    LoadTexture(pathTexture);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //  Transmiterea variabilelor uniforme pentru MATRICEA DE TR
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[
    //  Transmiterea variabilei uniforme pentru TEXTURARE spre s
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0)
    glDrawArrays(GL_POLYGON, startPoint, 4);
}
//  Functia de desenarea a graficii pe ecran;
void RenderFunction(void)
{
    updateTranslationCoordonates();

    glClear(GL_COLOR_BUFFER_BIT);           //  Se curata ecranu
    //myMatrix = resizeMatrix;
    CreateVBO();
    //Background
    BuildObject(translCoordBackground, matrTranslBackground, "ba
    BuildObject(translCoordBackground, matrTranslBackground, "ba

    //Masina verde
    BuildObject(translCoordCarGreen1, matrTranslGreen1Car, "gree

    //Mașina rosie
    BuildObject(translCoordCarRed, matrTranslRedCar, "redCar.png

    //Mașina galbena
    BuildObject(translCoordCarYellow, matrTranslYellowCar, "yell
```

```
    //Mașina verde
    BuildObject(translCoordCarGreen2, matrTranslGreen2Car, "gree



    //EndGame
    if (GameOver == true) {
        BuildObject(translCoordGameOver, matrTranslGameOver, "Ga
        BuildObject(translCoordCollision, matrTranslCollision, "

    }



    glutSwapBuffers();
    glFlush();
}

//  Punctul de intrare in program, se ruleaza rutina OpenGL;
int main(int argc, char* argv[])
{
    //  Se initializeaza GLUT si contextul OpenGL si se configur

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(winWidth, winHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Utilizare glm::ortho - OpenGL <<nou>>");

    //  Se initializeaza GLEW si se verifica suportul de extensi
    //  Trebuie initializat inainte de desenare;

    glewInit();

    Initialize();                       //  Setarea parametrilor
    glutDisplayFunc(RenderFunction);    //  Desenarea scenei in
    glutSpecialFunc(KeysPressed);       //  Callback cand tasta
    glutSpecialUpFunc(KeysReleased);    //  Callback cand tasta
```

```
    glutCloseFunc(Cleanup);             //  Eliberarea resurselo

    //  Bucla principala de procesare a evenimentelor GLUT (func
    //  Prelucreaza evenimentele si deseneaza fereastra OpenGL p

    glutMainLoop();

    return 0;
}
```

Shader.frag

```
// ================================================
// | Grafica pe calculator                        |
// ================================================
//  Shaderul de fragment / Fragment shader - afecteaza culoarea
//

#version 330 core

//  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;

//  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);
   out_Color = mix(texture(myTexture, tex_Coord), ex_Color, 0.2)
  }
```

Shader.vert

```
//
// ================================================
// | Grafica pe calculator                        |
// ================================================
//  Shaderul de varfuri / Vertex shader - afecteaza geometria sc
//

#version 330 core

//  Variabile de intrare (dinspre programul principal);
layout (location = 0) in vec4 in_Position;     //  Se preia din
layout (location = 1) in vec4 in_Color;        //  Se preia din
layout (location=2) in vec2 texCoord;          //  Se preia din

//  Variabile de iesire;
out vec4 gl_Position;   //  Transmite pozitia actualizata spre p
out vec4 ex_Color;      //  Transmite culoarea (de modificat in
out vec2 tex_Coord;     //  Transmite textura (de modificat in S

//  Variabile uniforme;
uniform mat4 myMatrix;

void main(void)
  {
    gl_Position = myMatrix*in_Position;
    ex_Color=in_Color;
```

```
     tex_Coord = vec2(texCoord.x, 1-texCoord.y);
    }
```