Documentație
Proiect 1

Echipa

Petrovici Ricardo - *351*
Mihalache Sebastian-Ștefan - *352*

Link Github: https://github.com/sebimih13/Grafica-Depasire

Conceptul proiectului

Acest proiect este o simulare interactivă care pune jucătorul în rolul unui șofer ce trebuie să depășească alte mașini aflate pe șosea fără a provoca accidente. Folosind un set simplu de controale, jucătorul trebuie să mențină o traiectorie sigură în timp ce depășește mașini aflate în mișcare pe o șosea cu două sensuri de mers.

Mașina jucătorului este fixată în stânga ecranului pentru a simula mișcarea continuă a drumului. Aceasta creează o iluzie de viteză și fluiditate în mișcare, iar jucătorul are o vizibilitate clară asupra drumului din față, oferindu-i suficient timp pentru a lua decizia dacă este posibilă sau nu, efectuarea depășirii în siguranță a mașinii din fața lui.

Șoseaua este prevăzută cu câte o bandă pe fiecare sens de circulație și este traversată de mașini care se deplasează cu o viteză constantă, atât în direcția jucătorului, cât și pe contrasens.

Ce transformari au fost incluse

Translația + Scalarea

- Aceste transformări sunt folosite atât în funcțiile pentru desenarea fundalului, cât și pentru desenarea mașinilor
- Pentru a reprezenta o pădure de copaci, desenăm mai mulți copaci folosind același VAO. Fiecare instanță a copacului este translatată și scalată diferit, ceea ce creează un efect vizual de diversitate.
- Tot la desenarea fundalului, am aplicat translații pentru a putea simula mișcarea continuă a mediului înconjurător față de poziția camerei. Când un obiect din fundal părăsește proiecția ortogonală, acesta este repoziționat din nou în partea dreaptă a ecranului.
- Calcularea coliziunilor.

Rotirea

- Această transformare este aplicată asupra mașinii controlate de jucător pentru a simula virajele la stânga și la dreapta, permițând schimbarea benzii de circulație.
- Pentru a roti mașinile care vin din sensul opus, simulând astfel comportamentul traficului pe ambele direcții.
- Calcularea coliziunilor.

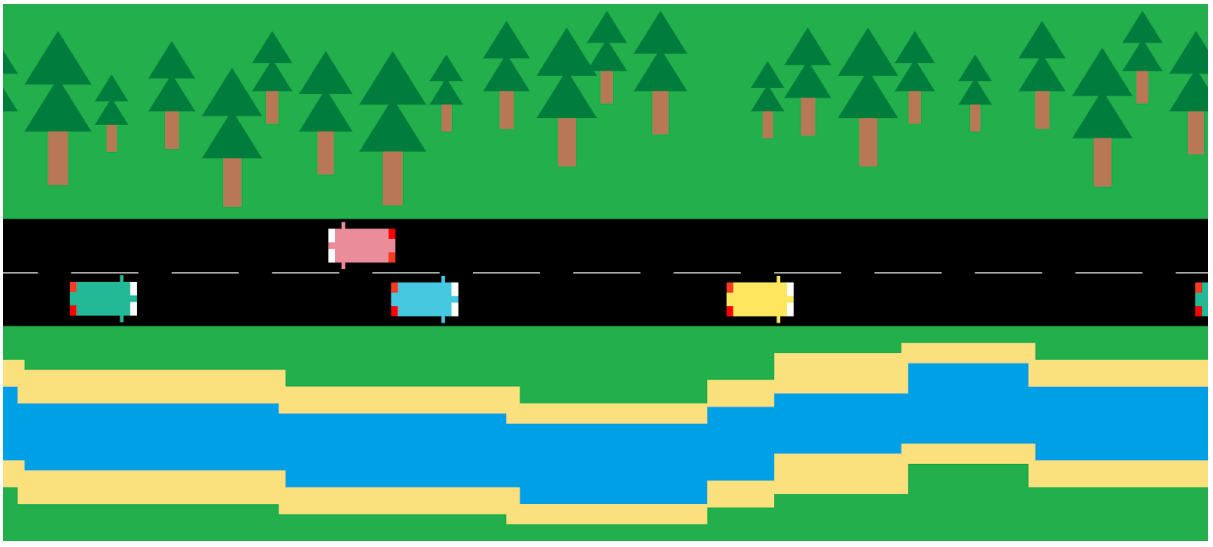
De ce este original

- **Animații dinamice**: Jucătorul primește un feedback vizual prin animații atunci când mașina execută viraje la stânga sau dreapta ori frânează. Aceste animații ajută la crearea unei experiențe mai realiste și mai intuitive pentru utilizator.

- **Trasee generate aleator**: Fiecare experiență de joc este diferită, deoarece mașinile care apar pe ambele sensuri de mers sunt generate aleator. Astfel, jucătorul nu va parcurge niciodată același traseu.

- **Mașini personalizate**: La generarea mașinilor, sunt utilizate diferite culori.

Capturi de ecran (cod, rezultat) relevante



Contributii individuale

Petrovici Ricardo
- sistemul de coliziune
- sistemul de input pentru mașina controlată de jucător
- sistemul de actualizare a mașinii (rotație + viteză)
- generarea mașinilor pe stradă

Mihalache Sebastian-Ștefan
- crearea fundalului și logica pentru translatarea acestuia în funcție de viteza mașinii
- sistemul de input pentru mașina controlată de jucător
- generarea mașinilor cu culori diferite
- animațiile de semnalizare și frânare pentru mașina jucătorului

Vertex Shader

```
#version 330 core

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

Fragment Shader

```
#version 330 core

in vec4 ex_Color;

out vec4 out_Color;

uniform int codColShader;
uniform int changeCarColor;
uniform vec4 newCarColor;

void main(void)
{
	if (changeCarColor == 1)
	{
		out_Color = newCarColor;
		return;
	}

	switch(codColShader) {
		case 0:
		{
			out_Color = ex_Color;
			break;
		}
		case 1:
		{
			out_Color = vec4(1.0, 1.0, 1.0, 1.0);
			break;
		}
		case 2: // headlights
		{
			out_Color = vec4(0.969, 0.969, 0.004, 1.0);
			break;
		}
		case 3: // break lights
		{
			out_Color = vec4(0.565, 0.0, 0.0, 1.0);
			break;
		}
		default:
		{
			out_Color = ex_Color;
			break;
		}
```

```
	}
}
```

Codul sursa

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <random>
#include <unordered_map>
#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"

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

constexpr GLuint winWidth = 1800, winHeight = 800;
constexpr GLfloat xMin = -(winWidth / 2.0f), xMax = (winWidth / 2.0f),
yMin = -(winHeight / 2.0f), yMax = (winHeight / 2.0f);

GLuint VaoIdBackground, VboIdBackground, EboIdBackground;
GLuint VaoIdCar, VboIdCar, EboIdCar;
GLuint VaoIdCars, VboIdCars, EboIdCars, codColLocation;
GLuint ProgramId;
GLuint myMatrixUniformLocation, changeCarColorUniformLocation,
newCarColorUniformLocation;

glm::mat4 resizeMatrix;

std::unordered_map<char, bool> keyStates;

float distance = 0;

void CreateVAOBackground()
{
	constexpr GLfloat Vertices[] = {
		// tree
		-50.0f, 0.0f, 0.0f, 1.0f,
```

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

        -50.0f, 70.0f, 0.0f, 1.0f,
        50.0f, 70.0f, 0.0f, 1.0f,
        0.0f, 150.0f, 0.0f, 1.0f,

        -15.0f, 0.0f, 0.0f, 1.0f,
        15.0f, 0.0f, 0.0f, 1.0f,
        15.0f, -80.0f, 0.0f, 1.0f,
        -15.0f, -80.0f, 0.0f, 1.0f,

        // highway
        xMin, 80.0f, 0.0f, 1.0f,
        xMax, 80.0f, 0.0f, 1.0f,
        xMax, -80.0f, 0.0f, 1.0f,
        xMin, -80.0f, 0.0f, 1.0f,

        // strip
        -50.0f, 0.0f, 0.0f, 1.0f,
        50.0f, 0.0f, 0.0f, 1.0f,

        // beach
        -1.0f, 1.0f, 0.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 1.0f,
        1.0f, -1.0f, 0.0f, 1.0f,
        -1.0f, -1.0f, 0.0f, 1.0f,

        // river
        -1.0f, 1.0f, 0.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 1.0f,
        1.0f, -1.0f, 0.0f, 1.0f,
        -1.0f, -1.0f, 0.0f, 1.0f
    };

    constexpr GLfloat Colors[] = {
        // tree
        0.0f, 0.502f, 0.251f, 1.0f,
        0.0f, 0.502f, 0.251f, 1.0f,
        0.0f, 0.502f, 0.251f, 1.0f,

        0.0f, 0.502f, 0.251f, 1.0f,
        0.0f, 0.502f, 0.251f, 1.0f,
        0.0f, 0.502f, 0.251f, 1.0f,

        0.729f, 0.478f, 0.341f, 1.0f,
```

```
        0.729f, 0.478f, 0.341f, 1.0f,
        0.729f, 0.478f, 0.341f, 1.0f,
        0.729f, 0.478f, 0.341f, 1.0f,

        // highway
        0.0f, 0.0f, 0.0f, 1.0f,
        0.0f, 0.0f, 0.0f, 1.0f,
        0.0f, 0.0f, 0.0f, 1.0f,
        0.0f, 0.0f, 0.0f, 1.0f,

        // strip
        1.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 1.0f, 1.0f,

        // beach
        0.996f, 0.886f, 0.49f, 1.0f,
        0.996f, 0.886f, 0.49f, 1.0f,
        0.996f, 0.886f, 0.49f, 1.0f,
        0.996f, 0.886f, 0.49f, 1.0f,

        // river
        0.0f, 0.639f, 0.91f, 1.0f,
        0.0f, 0.639f, 0.91f, 1.0f,
        0.0f, 0.639f, 0.91f, 1.0f,
        0.0f, 0.639f, 0.91f, 1.0f
    };

    constexpr GLuint Indices[] = {
        // tree
        0, 1, 2,
        3, 4, 5,
        6, 7, 8,
        6, 8, 9,

        // highway
        10, 11, 12,
        10, 12, 13,

        // strip
        14, 15,

        // beach
        16, 17, 18, 19,

        // river
        20, 21, 22, 23
```

```
	};

	glGenVertexArrays(1, &VaoIdBackground);
	glBindVertexArray(VaoIdBackground);

	// Vertex Buffer
	glGenBuffers(1, &VboIdBackground);
	glBindBuffer(GL_ARRAY_BUFFER, VboIdBackground);
	glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices) + sizeof(Colors),
NULL, GL_STATIC_DRAW);
	glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices), Vertices);
	glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices), sizeof(Colors),
Colors);

	glGenBuffers(1, &EboIdBackground);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboIdBackground);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices,
GL_STATIC_DRAW);

	// atribut 0 => (location = 0)
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);

	// atribut 1 => (location = 1)
	glEnableVertexAttribArray(1);
	glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, (const
GLvoid*)sizeof(Vertices));
}

void CreateVAOCar()
{
	GLfloat Vertices[] = {
		// car
		-50.0f, 25.0f, 0.0f, 1.0f,
		50.0f, 25.0f, 0.0f, 1.0f,
		50.0f, -25.0f, 0.0f, 1.0f,
		-50.0f, -25.0f, 0.0f, 1.0f,

		// brake lights
		-50.0f, 25.0f, 0.0f, 1.0f,
		-40.0f, 25.0f, 0.0f, 1.0f,
		-40.0f, 10.0f, 0.0f, 1.0f,
		-50.0f, 10.0f, 0.0f, 1.0f,

		-50.0f, -25.0f, 0.0f, 1.0f,
		-50.0f, -10.0f, 0.0f, 1.0f,
```

```
        -40.0f, -10.0f, 0.0f, 1.0f,
        -40.0f, -25.0f, 0.0f, 1.0f,

        // headlights
        50.0f, 25.0f, 0.0f, 1.0f,
        40.0f, 25.0f, 0.0f, 1.0f,
        40.0f, 5.0f, 0.0f, 1.0f,
        50.0f, 5.0f, 0.0f, 1.0f,

        50.0f, -25.0f, 0.0f, 1.0f,
        50.0f, -5.0f, 0.0f, 1.0f,
        40.0f, -5.0f, 0.0f, 1.0f,
        40.0f, -25.0f, 0.0f, 1.0f,

        // side-view mirror
        30.0f, 25.0f, 0.0f, 1.0f,
        30.0f, 35.0f, 0.0f, 1.0f,
        25.0f, 35.0f, 0.0f, 1.0f,
        25.0f, 25.0f, 0.0f, 1.0f,

        30.0f, -25.0f, 0.0f, 1.0f,
        30.0f, -35.0f, 0.0f, 1.0f,
        25.0f, -35.0f, 0.0f, 1.0f,
        25.0f, -25.0f, 0.0f, 1.0f
    };

    GLfloat Colors[] = {
        // car
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,

        // brake lights
        0.969f, 0.243f, 0.133f, 1.0f,
        0.969f, 0.243f, 0.133f, 1.0f,
        0.969f, 0.243f, 0.133f, 1.0f,
        0.969f, 0.243f, 0.133f, 1.0f,

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

        // headlights
        1.0f, 1.0f, 1.0f, 1.0f,
```

```
        1.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 1.0f, 1.0f,

        1.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 1.0f, 1.0f,

        // Side-view mirror
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,

        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f
    };

    GLuint Indices[] = {
        // car
        0, 1, 2, 3,

        // Side-view mirror
        20, 21, 22, 23,
        24, 25, 26, 27,

        // brake lights
        4, 5, 6, 7,
        8, 9, 10, 11,

        // headlights
        12, 13, 14, 15,
        16, 17, 18, 19
    };

    glGenVertexArrays(1, &VaoIdCar);
    glBindVertexArray(VaoIdCar);

    // Vertex Buffer
    GLuint VboIdCar;
    glGenBuffers(1, &VboIdCar);
    glBindBuffer(GL_ARRAY_BUFFER, VboIdCar);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices) + sizeof(Colors),
```

```
NULL, GL_STATIC_DRAW);
	glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices), Vertices);
	glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices), sizeof(Colors),
Colors);

	glGenBuffers(1, &EboIdCar);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboIdCar);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices,
GL_STATIC_DRAW);

	// atribut 0 => (location = 0)
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);

	// atribut 1 => (location = 1)
	glEnableVertexAttribArray(1);
	glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, (const
GLvoid*)sizeof(Vertices));
}

void DestroyVAOs(void)
{
	glDisableVertexAttribArray(1);
	glDisableVertexAttribArray(0);

	glBindBuffer(GL_ARRAY_BUFFER, 0);
	glDeleteBuffers(1, &EboIdBackground);
	glDeleteBuffers(1, &VboIdBackground);
	glDeleteBuffers(1, &EboIdCars);
	glDeleteBuffers(1, &VboIdCars);
	glDeleteBuffers(1, &EboIdCar);
	glDeleteBuffers(1, &VboIdCar);

	glBindVertexArray(0);
	glDeleteVertexArrays(1, &VaoIdBackground);
	glDeleteVertexArrays(1, &VaoIdCars);
	glDeleteVertexArrays(1, &VaoIdCar);
}

void CreateShaders(void)
{
	ProgramId = LoadShaders("example.vert", "example.frag");
	glUseProgram(ProgramId);
}

void DestroyShaders(void)
```

```
{
	glDeleteProgram(ProgramId);
}

void Initialize(void)
{
	glClearColor(0.137f, 0.694f, 0.302f, 1.0f);

	CreateVAOBackground();
	CreateVAOCar();
	CreateShaders();

	myMatrixUniformLocation = glGetUniformLocation(ProgramId,
"myMatrix");
	codColLocation = glGetUniformLocation(ProgramId, "codColShader");

	changeCarColorUniformLocation = glGetUniformLocation(ProgramId,
"changeCarColor");
	newCarColorUniformLocation = glGetUniformLocation(ProgramId,
"newCarColor");

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

void RenderBackground() {
	glBindVertexArray(VaoIdBackground);

	// trees
	{
		std::vector<std::pair<GLfloat, GLfloat>> position = {
			std::make_pair(-500.0f, 250.0f),
			std::make_pair(-400.0f, 270.0f),
			std::make_pair(-310.0f, 200.0f),
			std::make_pair(-250.0f, 300.0f),
			std::make_pair(-170.0f, 240.0f),
			std::make_pair(-70.0f, 210.0f),

			std::make_pair(10.0f, 220.0f),
			std::make_pair(100.0f, 240.0f),
			std::make_pair(190.0f, 170.0f),
			std::make_pair(250.0f, 270.0f),
			std::make_pair(330.0f, 210.0f),
			std::make_pair(430.0f, 180.0f),

			std::make_pair(510.0f, 250.0f),
			std::make_pair(600.0f, 270.0f),
```

```
                std::make_pair(690.0f, 230.0f),
                std::make_pair(750.0f, 300.0f),
                std::make_pair(830.0f, 270.0f),

                std::make_pair(-810.0f, 240.0f),
                std::make_pair(-750.0f, 260.0f),
                std::make_pair(-660.0f, 230.0f),
                std::make_pair(-590.0f, 270.0f)
        };

        std::vector<std::pair<GLfloat, GLfloat>> scale = {
                std::make_pair(0.5f, 0.5f),
                std::make_pair(0.7f, 0.7f),
                std::make_pair(0.9f, 0.9f),
                std::make_pair(0.6f, 0.6f),
                std::make_pair(0.8f, 0.8f),
                std::make_pair(1.0f, 1.0f),

                std::make_pair(0.5f, 0.5f),
                std::make_pair(0.7f, 0.7f),
                std::make_pair(0.9f, 0.9f),
                std::make_pair(0.6f, 0.6f),
                std::make_pair(0.8f, 0.8f),
                std::make_pair(1.0f, 1.0f),

                std::make_pair(0.5f, 0.5f),
                std::make_pair(0.7f, 0.7f),
                std::make_pair(0.9f, 0.9f),
                std::make_pair(0.6f, 0.6f),
                std::make_pair(0.8f, 0.8f),

                std::make_pair(0.5f, 0.5f),
                std::make_pair(0.7f, 0.7f),
                std::make_pair(0.9f, 0.9f),
                std::make_pair(0.6f, 0.6f),
                std::make_pair(0.8f, 0.8f)
        };

        for (int i = 0; i < position.size() && i < scale.size();
i++)
        {
                GLfloat posX = position[i].first - distance;
                GLfloat posY = position[i].second;

                const GLfloat& scaleX = scale[i].first;
                const GLfloat& scaleY = scale[i].second;
```

```
                if (posX - scaleX < xMin)
                {
                        glm::mat4 traslateMatrix =
glm::translate(glm::mat4(1.0f), glm::vec3(posX, posY, 1.0f));
                        glm::mat4 scaleMatrix =
glm::scale(glm::mat4(1.0f), glm::vec3(scaleX, scaleY, 1.0f));
                        glm::mat4 myMatrix = resizeMatrix *
traslateMatrix * scaleMatrix;
                        glUniformMatrix4fv(myMatrixUniformLocation, 1,
GL_FALSE, &myMatrix[0][0]);

                        glDrawElements(GL_TRIANGLES, 12, GL_UNSIGNED_INT,
(void*)(0));

                        float overflow = xMin - posX;
                        posX = xMax - overflow;
                }

                glm::mat4 traslateMatrix =
glm::translate(glm::mat4(1.0f), glm::vec3(posX, posY, 1.0f));
                glm::mat4 scaleMatrix = glm::scale(glm::mat4(1.0f),
glm::vec3(scaleX, scaleY, 1.0f));
                glm::mat4 myMatrix = resizeMatrix * traslateMatrix *
scaleMatrix;
                glUniformMatrix4fv(myMatrixUniformLocation, 1,
GL_FALSE, &myMatrix[0][0]);

                glDrawElements(GL_TRIANGLES, 12, GL_UNSIGNED_INT,
(void*)(0));
            }
      }

      { // highway
            glm::mat4 myMatrix = resizeMatrix;
            glUniformMatrix4fv(myMatrixUniformLocation, 1, GL_FALSE,
&myMatrix[0][0]);
      }
      glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (void*)(12 *
sizeof(GLuint)));

      // strip
      for (float posX = xMin; posX <= xMax; posX += 150.0f)
      {
            float newPosX = posX - distance;
```

```
            if (newPosX - 50.0f < xMin)
            {
                  glm::mat4 translateMatrix =
glm::translate(glm::mat4(1.0f), glm::vec3(newPosX, 0.0f, 1.0f));
                  glm::mat4 myMatrix = resizeMatrix * translateMatrix;
                  glUniformMatrix4fv(myMatrixUniformLocation, 1,
GL_FALSE, &myMatrix[0][0]);

                  glDrawElements(GL_LINES, 2, GL_UNSIGNED_INT,
(void*)(18 * sizeof(GLuint)));

                  float overflow = xMin - newPosX;
                  newPosX = xMax - overflow;
            }

            glm::mat4 translateMatrix = glm::translate(glm::mat4(1.0f),
glm::vec3(newPosX, 0.0f, 1.0f));
            glm::mat4 myMatrix = resizeMatrix * translateMatrix;
            glUniformMatrix4fv(myMatrixUniformLocation, 1, GL_FALSE,
&myMatrix[0][0]);

            glDrawElements(GL_LINES, 2, GL_UNSIGNED_INT, (void*)(18 *
sizeof(GLuint)));
      }

      { // river
            std::vector<std::pair<GLfloat, GLfloat>> position = {
                  std::make_pair(-850, -240.0f),
                  std::make_pair(-700, -210.0f),
                  std::make_pair(-510, -180.0f),
                  std::make_pair(-270, -210.0f),
                  std::make_pair(70, -230.0f),
                  std::make_pair(440, -250.0f),
                  std::make_pair(750, -270.0f)
            };

            std::vector<std::pair<GLfloat, GLfloat>> scale = {
                  std::make_pair(50.0f, 80.0f),
                  std::make_pair(100.0f, 90.0f),
                  std::make_pair(100.0f, 75.0f),
                  std::make_pair(150.0f, 80.0f),
                  std::make_pair(200.0f, 85.0f),
                  std::make_pair(180.0f, 80.0f),
                  std::make_pair(150.0f, 75.0f)
            };
```

```
            for (int i = 0; i < position.size() && i < scale.size();
i++)
            {
                  GLfloat posX = position[i].first - distance;
                  GLfloat posY = position[i].second;

                  const GLfloat& scaleX = scale[i].first;
                  const GLfloat& scaleY = scale[i].second;

                  if (posX - scaleX < xMin)
                  {
                        glm::mat4 traslateMatrix =
glm::translate(glm::mat4(1.0f), glm::vec3(posX, posY, 1.0f));
                        glm::mat4 scaleMatrix =
glm::scale(glm::mat4(1.0f), glm::vec3(scaleX, scaleY, 1.0f));
                        glm::mat4 myMatrix = resizeMatrix *
traslateMatrix * scaleMatrix;
                        glUniformMatrix4fv(myMatrixUniformLocation, 1,
GL_FALSE, &myMatrix[0][0]);

                        glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT,
(void*)(24 * sizeof(GLuint)));

                        float overflow = xMin - posX;
                        posX = xMax - overflow;
                  }

                  glm::mat4 traslateMatrix =
glm::translate(glm::mat4(1.0f), glm::vec3(posX, posY, 1.0f));
                  glm::mat4 scaleMatrix = glm::scale(glm::mat4(1.0f),
glm::vec3(scaleX, scaleY, 1.0f));
                  glm::mat4 myMatrix = resizeMatrix * traslateMatrix *
scaleMatrix;
                  glUniformMatrix4fv(myMatrixUniformLocation, 1,
GL_FALSE, &myMatrix[0][0]);

                  glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT,
(void*)(24 * sizeof(GLuint)));
            }
      }

      // beach
      {
            std::vector<std::pair<GLfloat, GLfloat>> position = {
                  std::make_pair(-850, -180.0f),
                  std::make_pair(-700, -150.0f),
```

```
                std::make_pair(-510, -120.0f),
                std::make_pair(-270, -150.0f),
                std::make_pair(70, -170.0f),
                std::make_pair(440, -190.0f),
                std::make_pair(750, -210.0f),

                std::make_pair(-850, -330.0f),
                std::make_pair(-700, -300.0f),
                std::make_pair(-510, -270.0f),
                std::make_pair(-270, -300.0f),
                std::make_pair(70, -320.0f),
                std::make_pair(440, -340.0f),
                std::make_pair(750, -360.0f),
        };

        std::vector<std::pair<GLfloat, GLfloat>> scale = {
                std::make_pair(50.0f, 20.0f),
                std::make_pair(100.0f, 30.0f),
                std::make_pair(100.0f, 15.0f),
                std::make_pair(150.0f, 20.0f),
                std::make_pair(200.0f, 25.0f),
                std::make_pair(180.0f, 20.0f),
                std::make_pair(150.0f, 15.0f),

                std::make_pair(50.0f, 20.0f),
                std::make_pair(100.0f, 30.0f),
                std::make_pair(100.0f, 15.0f),
                std::make_pair(150.0f, 20.0f),
                std::make_pair(200.0f, 25.0f),
                std::make_pair(180.0f, 20.0f),
                std::make_pair(150.0f, 15.0f),
        };

        for (int i = 0; i < position.size() && i < scale.size();
i++)
        {
                GLfloat posX = position[i].first - distance;
                GLfloat posY = position[i].second;

                const GLfloat& scaleX = scale[i].first;
                const GLfloat& scaleY = scale[i].second;

                if (posX - scaleX < xMin)
                {
                        glm::mat4 traslateMatrix =
glm::translate(glm::mat4(1.0f), glm::vec3(posX, posY, 1.0f));
```

```
                        glm::mat4 scaleMatrix =
glm::scale(glm::mat4(1.0f), glm::vec3(scaleX, scaleY, 1.0f));
                        glm::mat4 myMatrix = resizeMatrix *
traslateMatrix * scaleMatrix;
                        glUniformMatrix4fv(myMatrixUniformLocation, 1,
GL_FALSE, &myMatrix[0][0]);

                        glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT,
(void*)(20 * sizeof(GLuint)));

                        float overflow = xMin - posX;
                        posX = xMax - overflow;
                  }

                  glm::mat4 traslateMatrix =
glm::translate(glm::mat4(1.0f), glm::vec3(posX, posY, 1.0f));
                  glm::mat4 scaleMatrix = glm::scale(glm::mat4(1.0f),
glm::vec3(scaleX, scaleY, 1.0f));
                  glm::mat4 myMatrix = resizeMatrix * traslateMatrix *
scaleMatrix;
                  glUniformMatrix4fv(myMatrixUniformLocation, 1,
GL_FALSE, &myMatrix[0][0]);

                  glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT,
(void*)(20 * sizeof(GLuint)));
            }
      }
}

class Car {
public:
      glm::vec4 color;
      float carPozX;
      float carPozY;
      float turningAngle = 0.0f;
      bool OppositeDirection;
      bool leftTurn = false, rightTurn = false, brake = false;

public:
      Car(float carPozX, float carPozY,  bool Opposit = false) {
            OppositeDirection = Opposit;
            if (Opposit) turningAngle = 180.0f;
            color = getRandomCarColor();
            this->carPozX = carPozX;
            this->carPozY = carPozY;
      }
```

```
	glm::vec4 getRandomCarColor()
	{
		static const std::vector<glm::vec4> colors = {
			glm::vec4(0.502f, 0.612f, 0.075f, 1.0f),
			glm::vec4(0.925f, 0.925f, 0.639f, 1.0f),
			glm::vec4(1.0f, 0.918f, 0.38f, 1.0f),
			glm::vec4(0.929f, 0.549f, 0.616f, 1.0f),
			glm::vec4(0.282f, 0.792f, 0.894f, 1.0f),
			glm::vec4(1.0f, 0.455f, 0.0f, 1.0f),
			glm::vec4(0.149f, 0.737f, 0.6f, 1.0f)
		};

		static std::random_device rd;
		static std::mt19937 gen(rd());
		std::uniform_int_distribution<int> dist(0, colors.size() -
1);
		int randomIndex = dist(gen);
		return colors[randomIndex];
	}

	void drawCar() {
		glm::mat4 myMatrix = resizeMatrix *
glm::translate(glm::mat4(1.0f), glm::vec3(carPozX, carPozY, 0))
                                                        *
glm::rotate(glm::mat4(1.0f), glm::radians(turningAngle), glm::vec3(0.0f,
0.0f, 1.0f));
		glUniformMatrix4fv(myMatrixUniformLocation, 1, GL_FALSE,
&myMatrix[0][0]);
		glUniform1i(changeCarColorUniformLocation, 1);
		glUniform4fv(newCarColorUniformLocation, 1, &color[0]);

		glDrawElements(GL_QUADS, 12, GL_UNSIGNED_INT, (void*)(0));

		glUniform1i(changeCarColorUniformLocation, 0);

		int codCol = 0;
		if (brake)
		{
			codCol = 3;
		}
		glUniform1i(codColLocation, codCol);
		glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, (void*)(12 *
sizeof(GLuint)));
		glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, (void*)(16 *
sizeof(GLuint)));
```

```
            codCol = 0;
            if (leftTurn)
            {
                  codCol = 2;
            }
            glUniform1i(codColLocation, codCol);
            glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, (void*)(20 *
sizeof(GLuint)));

            codCol = 0;
            if (rightTurn)
            {
                  codCol = 2;
            }
            glUniform1i(codColLocation, codCol);
            glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, (void*)(24 *
sizeof(GLuint)));

            codCol = 0;
            glUniform1i(codColLocation, codCol);
      }
};

Car drivenCar(-750.0f, 0.0f);
std::vector<Car> generatedCars;

float speedDrivenCar = 1.5f;
const int timer = 16;
int codCol;
float lastGeneratedCar = 0, lastGeneratedCarOpposite = 0;
bool colide = false;

void resetState() {
      generatedCars.clear();
      drivenCar.carPozX = -750.0f;
      drivenCar.carPozY = 0.0f;
      drivenCar.turningAngle = 0.0f;
      drivenCar.color = drivenCar.getRandomCarColor();
      speedDrivenCar = 1.5f;
      codCol = 0;
      colide = false;
}

bool checkPointInRectangle(float px, float py, float xMin, float xMax,
float yMin, float yMax) {
```

```
	if (xMin <= px && px <= xMax && yMin <= py && py <= yMax) return
true;
	return false;
}

void checkColision() {
	float newLowerLeftX = drivenCar.carPozX - 50.0f *
cos(glm::radians(drivenCar.turningAngle)) + 25.0f *
sin(glm::radians(drivenCar.turningAngle));
	float newLowerLeftY = drivenCar.carPozY - 25.0f *
cos(glm::radians(drivenCar.turningAngle)) - 50.0f *
sin(glm::radians(drivenCar.turningAngle));

	float newUpperLeftX = drivenCar.carPozX - 50.0f *
cos(glm::radians(drivenCar.turningAngle)) - 25.0f *
sin(glm::radians(drivenCar.turningAngle));
	float newUpperLeftY = drivenCar.carPozY + 25.0f *
cos(glm::radians(drivenCar.turningAngle)) - 50.0f *
sin(glm::radians(drivenCar.turningAngle));

	float newUpperRightX = drivenCar.carPozX + 50.0f *
cos(glm::radians(drivenCar.turningAngle)) - 25.0f *
sin(glm::radians(drivenCar.turningAngle));
	float newUpperRightY = drivenCar.carPozY + 25.0f *
cos(glm::radians(drivenCar.turningAngle)) + 50.0f *
sin(glm::radians(drivenCar.turningAngle));

	float newLowerRightX = drivenCar.carPozX + 50.0f *
cos(glm::radians(drivenCar.turningAngle)) + 25.0f *
sin(glm::radians(drivenCar.turningAngle));
	float newLowerRightY = drivenCar.carPozY - 25.0f *
cos(glm::radians(drivenCar.turningAngle)) + 50.0f *
sin(glm::radians(drivenCar.turningAngle));



	for (auto x : generatedCars) {
		float XMin = -50 + x.carPozX, XMax = XMin + 100;
		float YMin = -25 + x.carPozY, YMax = YMin + 50;

		colide |= checkPointInRectangle(newLowerLeftX,
newLowerLeftY, XMin, XMax, YMin, YMax);
		colide |= checkPointInRectangle(newUpperLeftX,
newUpperLeftY, XMin, XMax, YMin, YMax);
		colide |= checkPointInRectangle(newUpperRightX,
newUpperRightY, XMin, XMax, YMin, YMax);
```

```
            colide |= checkPointInRectangle(newLowerRightX,
newLowerRightY, XMin, XMax, YMin, YMax);

      };

      if (newLowerLeftY < -85.0f || newUpperLeftY < -85.0f ||
newUpperRightY < -85.0f || newLowerRightY < -85.0f) colide = true;
      if (newLowerLeftY > 85.0f || newUpperLeftY > 85.0f ||
newUpperRightY > 85.0f || newLowerRightY > 85.0f) colide = true;
}

void recalcSpeed(unsigned int lastSpeedKey) {
      switch (lastSpeedKey) {
      case 'w':
            speedDrivenCar += 0.0035f * (15.0f - speedDrivenCar);
            break;
      case 's':
            speedDrivenCar -= 0.18f;
            break;
      default:
            speedDrivenCar -= 0.005f;
            break;
      }

      if (speedDrivenCar > 15.0f) speedDrivenCar = 15.0f;
      if (speedDrivenCar < 1.0f) speedDrivenCar = 1.0f;
}

void recalcAngle() {
      if (drivenCar.turningAngle > 0) drivenCar.turningAngle -=
glm::min(0.5f, drivenCar.turningAngle);
      else drivenCar.turningAngle -= glm::max(-0.5f,
drivenCar.turningAngle);
      if (drivenCar.turningAngle > 45) drivenCar.turningAngle = 45;
      if (drivenCar.turningAngle < -45) drivenCar.turningAngle = -45;
}


void updatePosition() {
      for (auto& x : generatedCars) {
            x.carPozX -= speedDrivenCar;
            if (x.OppositeDirection) x.carPozX -= 1.5f;
            else x.carPozX += 1.5f;
      }

      lastGeneratedCar += speedDrivenCar - 1.5f;
```

```
	lastGeneratedCarOpposite += speedDrivenCar + 1.5f;
}

void eraseCars() {
	for (int i = 0; i < generatedCars.size(); i++) {
		if (generatedCars[i].carPozX <= -950.0f) {
			generatedCars.erase(generatedCars.begin() + i);
			i--;
		}
	}
}

float generateRandomFloat() {
	static std::random_device rd;
	static std::mt19937 gen(rd());
	std::uniform_real_distribution<float> dist(0.0f, 1.0f);
	return dist(gen);
}

void generateCars() {
	if (lastGeneratedCar > 400.0f) {
		float value = generateRandomFloat();
		if (value < 0.15f) {
			generatedCars.push_back(Car(950.0f, -40.0f, false));
			lastGeneratedCar = 0.0f;
		}
		else lastGeneratedCar -= 100.0f;
	}

	if (lastGeneratedCarOpposite > 400.0f) {
		float value = generateRandomFloat();
		if (value < 0.05f) {
			generatedCars.push_back(Car(950.0f, 40.0f, true));
			lastGeneratedCarOpposite = 0.0f;
		}
		else lastGeneratedCarOpposite -= 100.0f;
	}
}

void drawCars() {
	for (auto x : generatedCars) {
		x.drawCar();
	}
}

void idleFunction(int val) {
```

```
	checkColision();
	if (colide) {
		codCol += 1;
		codCol %= 2;
		glutPostRedisplay();
		glutTimerFunc(500, idleFunction, 0);
		return;
	}

	updatePosition();
	eraseCars();
	generateCars();

	// input
	if (keyStates['w'] || keyStates['W'])
	{
		recalcSpeed('w');
	}

	if (keyStates['s'] || keyStates['S'])
	{
		recalcSpeed('s');
	}

	if (keyStates['a'] || keyStates['A'])
	{
		if (!colide) drivenCar.turningAngle += 1.5f;
	}

	if (keyStates['d'] || keyStates['D'])
	{
		if (!colide) drivenCar.turningAngle -= 1.5f;
	}

	recalcAngle();
	drivenCar.carPozY += speedDrivenCar *
sin(glm::radians(drivenCar.turningAngle));
	glutPostRedisplay();
	glutTimerFunc(timer, idleFunction, 0);

	distance += speedDrivenCar;
	if (distance > winWidth)
	{
		distance -= winWidth;
	}
}
```

```
void keyBoardFunc(unsigned char key, int x, int y)
{
	keyStates[key] = true;

	if (key == 'r' || key == 'R')
	{
		resetState();
	}

	switch (key)
	{
		case 'a': drivenCar.leftTurn  = true;	break;
		case 'A': drivenCar.leftTurn  = true;	break;
		case 'd': drivenCar.rightTurn = true;	break;
		case 'D': drivenCar.rightTurn = true;	break;
		case 's': drivenCar.brake       = true;	break;
		case 'S': drivenCar.brake       = true;	break;
	}
}

void keyBoardUpFunc(unsigned char key, int x, int y)
{
	keyStates[key] = false;

	switch (key)
	{
	case 'a': drivenCar.leftTurn  = false;	break;
	case 'A': drivenCar.leftTurn  = false;	break;
	case 'd': drivenCar.rightTurn = false;	break;
	case 'D': drivenCar.rightTurn = false;	break;
	case 's': drivenCar.brake     = false;	break;
	case 'S': drivenCar.brake     = false;	break;
	}
}


void RenderCars() {
	glBindVertexArray(VaoIdCar);
	drivenCar.drawCar();
	drawCars();

}

void RenderFunction(void)
```

```
{
	glClear(GL_COLOR_BUFFER_BIT);
	RenderBackground();
	RenderCars();

	glFlush();
}

void Cleanup(void)
{
	DestroyShaders();
	DestroyVAOs();
}

int main(int argc, char* argv[])
{
	glutInit(&argc, argv);

	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowPosition(50, 100);
	glutInitWindowSize(winWidth, winHeight);
	glutCreateWindow("Depasire");

	glewInit();
	Initialize();
	glutDisplayFunc(RenderFunction);
	glutTimerFunc(timer, idleFunction, 0);
	glutKeyboardFunc(keyBoardFunc);
	glutKeyboardUpFunc(keyBoardUpFunc);
	glutCloseFunc(Cleanup);
	glutMainLoop();
}
```