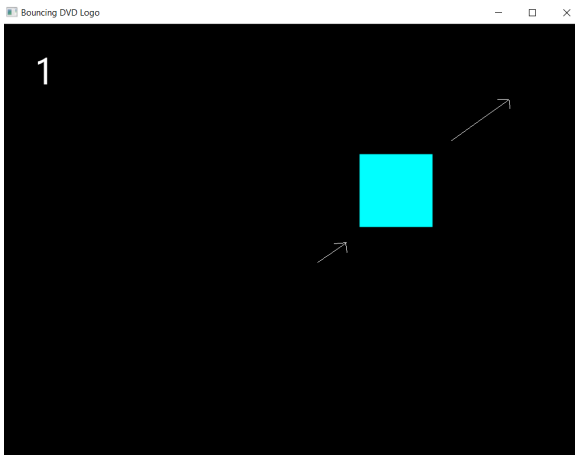
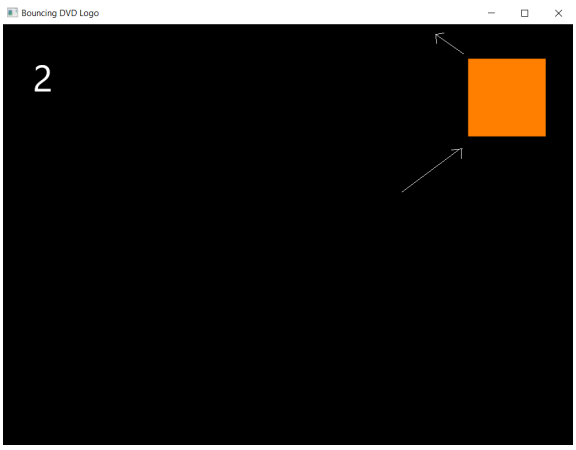
Proiect Grafica

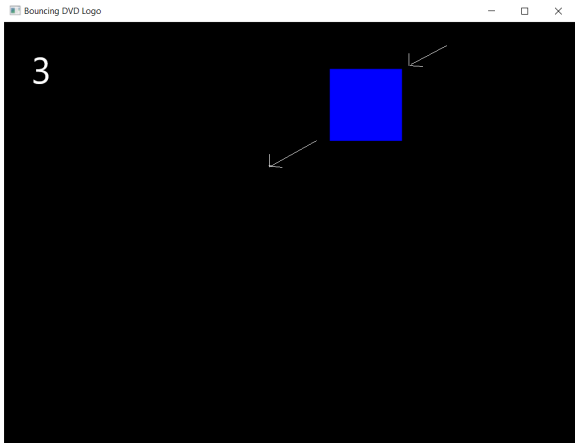
Realizat de: Erol Cherim si Teodora Lazaroiu

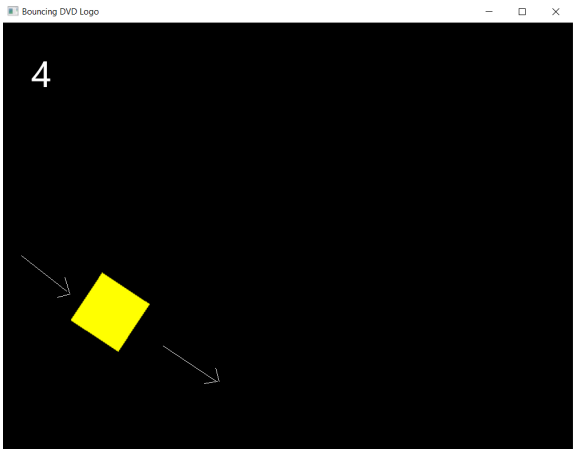
Descriere

Animatia este inspirata din scena screensaver-ului de la DVD playere. Patratul se misca in interiorul consolei iar cand se loveste de o margine isi modifica traiectoria, isi schimba culoarea si adauga o actiune sa scalare sau rotire. Pentru a porni animatia trebuie apasat click-ul de la mouse. Miscarea se realizeaza astfel:









Miscarea

Functia `move()` ofera valori pentru matricea de translatie `matrTransl` ce misca patratul intre marginile ferestrei. Bound-urile pana la care poate ajunge patratul sunt calculate in functia `resize()` in functie de dimensiunea ferestrei si a patratului.

```
void resize(void)
{
  positionXMax = windowWidth - logoWidth * i / 2;
  positionXMin = -windowWidth + logoWidth * i / 2;
  positionYMax = windowHeight - logoHeight  * i / 2;
  positionYMin = -windowHeight + logoHeight * i / 2;
}
```

Variabilele `windowWidth` si `windowHeight` sunt setate la inceputul programului iar variabila **i** reprezinta coeficientul cu care este scalat patratul prin matricea de scalare `matrScale`. Se adauga sau se scade jumatate din latura patratului deoarece bound-ul este implicit calculat din centrul patratului.

Matrici

In program sunt folosite 4 matrici ce sunt inmultite pentru a ajunge la rezultatul final:

- `resizeMatrix` este folosita pentru a aduce scena desenata la dimensiunea standard din OpenGL de $[-1, 1] \times [-1, 1]$
- `matrTransl` misca patratul in functie de coordonatele **positionX** si **positionY** care sunt calculate in functia **move()**
- `matrScale` mareste sau micsoreaza dimensiunea patratului pe ambele axe in functie de coeficientul **i** care va lua valori intre $[0.8, 1.2]$

- `matrRot` aplica patratului o miscare de rotatie un functie de valoarea **angle** care este initial 0 si din care se scade o valoare **beta**

```
resizeMatrix = glm::ortho(-windowWidth, windowWidth, -windowHeight, windowHeight);
matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(positionX, positionY, 0.0));
matrScale = glm::scale(glm::mat4(1.0f), glm::vec3(i, i, 0.0));
matrRot = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0, 0.0, 1.0));

myMatrix = resizeMatrix * matrTransl * matrScale * matrRot;
```

Actiuni

Valorile folosite in matricile de scalare si rotatie sunt modificate in functie de variabila `codAction` care determina ce actiune urmeaza sa fie aplicata patratului

- *`codAction = 1`* va face patratul sa se mareasca treptat pana la o anumita dimensiune
- *`codAction = 2`* va face patratul sa se micsoreze treptat pana la o anumita dimensiune
- *`codAction = 3`* va aplica patratului o miscare de rotatie, modificand valoarea unghiului
- *`altfel, codAction = 0`* patratul nu va suferi nicio modificare

```
if (codAction == 1)
  {
    if (i < 1.2) i = i + 0.0001;
  }
  else if (codAction == 2)
  {
    if (i > 0.8) i = i - 0.0001;
  }

  if (codAction == 3)
  {
    angle = angle - beta;
  }
  else
  {
    angle = 0.0;
  }
```

Contributii

Ideea proiectului a fost discutata si implementata de amandoi in mod egal:

- Teodora Lazaroiu: desenarea spatiului, functia ce dicteaza translatia patratului, shaderul de schimbarea culorii, documentatie
- Erol Cherim: functiile pentru recalcularea bound-urilor, actiunile de scalare si rotatie, documentatie

Ideea a fost inspirata dintr-un episod din serialul american The Office

Codul sursa

shaderCuloare.frag

```
// Shader-ul de fragment / Fragment shader
 #version 330

in vec4 ex_Color;
uniform int codCuloare;

out vec4 out_Color;

void main(void)
{
  switch (codCuloare)
  {
  case 0:
    // turcoaz
    out_Color=vec4 (0.0, 1.0, 1.0, 0.0);
    break;
  case 1:
    // portocaliu
    out_Color=vec4 (1.0, 0.5, 0.0, 0.0);
    break;
  case 2:
    // albastru
    out_Color=vec4 (0.0, 0.0, 1.0, 0.0);
    break;
  case 3:
    // galben
    out_Color=vec4 (1.0, 1.0, 0.0, 0.0);
    break;
  case 4:
    // rosu
    out_Color=vec4 (1.0, 0.0, 0.0, 0.0);
    break;
  case 5:
    // roz
    out_Color=vec4 (1.0, 0.0, 1.0, 0.0);
    break;
  case 6:
    // mov
    out_Color=vec4 (0.5, 0.0, 1.0, 0.0);
    break;
  case 7:
```

```
    // negru
    out_Color=vec4 (0.0, 0.0, 0.0, 0.0);
    break;
  default:
    break;
  };
}
```

shaderMatrice.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;
}
```

main.cpp

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <iostream>
#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"

using namespace std;

GLuint VaoId, VboId, ColorBufferId, ProgramId, myMatrixLocation,
matrScaleLocation, matrRotlLocation, matrTranslLocation, codColLocation;

int codCol = 0, codAction = 0;
float PI = 3.141592, i = 1;
float windowWidth = 800, windowHeight = 600;
float logoWidth = 200, logoHeight = 200;
```

```
float positionX = 0.0, positionY = 0.0;
float positionXMax = windowWidth - logoWidth / 2;
float positionXMin = -windowWidth + logoWidth / 2;
float positionYMax = windowHeight - logoHeight / 2;
float positionYMin = -windowHeight + logoHeight / 2;
float xSpeed = 0.12, ySpeed = 0.06;
float angle = 0.0, beta = 0.002;
glm::mat4 myMatrix, resizeMatrix, matrTransl, matrScale, matrRot;

void resize(void)
{
  positionXMax = windowWidth - logoWidth * i / 2;
  positionXMin = -windowWidth + logoWidth * i / 2;
  positionYMax = windowHeight - logoHeight  * i / 2;
  positionYMin = -windowHeight + logoHeight * i / 2;
}

void move(void)
{
  positionX += xSpeed;
  positionY += ySpeed;
  resize();
  if (positionX > positionXMax) {
    positionX = positionXMax;
    xSpeed = -xSpeed;
    codCol = (codCol + 1) % 7;
    codAction = (codAction + 1) % 4;
  }
  if (positionX < positionXMin) {
    positionX = positionXMin;
    xSpeed = -xSpeed;
    codCol = (codCol + 1) % 7;
    codAction = (codAction + 1) % 4;
  }
  if (positionY > positionYMax) {
    positionY = positionYMax;
    ySpeed = -ySpeed;
    codCol = (codCol + 1) % 7;
    codAction = (codAction + 1) % 4;
  }
  if (positionY < positionYMin) {
    positionY = positionYMin;
    ySpeed = -ySpeed;
    codCol = (codCol + 1) % 7;
    codAction = (codAction + 1) % 4;
  }
  glutPostRedisplay();
}

void mouse(int button, int state, int x, int y)
{
  // miscarea va incepe cand facem click cu mouse-ul
  glutIdleFunc(move);
}

void CreateVBO(void)
{
  // varfurile pentru patrat
```

```
  GLfloat Vertices[] = {
     -100.0f, -100.0f, 0.0f, 1.0f,
    100.0f, -100.0f, 0.0f, 1.0f,
    100.0f,  100.0f, 0.0f, 1.0f,
     -100.0f,  100.0f, 0.0f, 1.0f,
  };

  // culorile varfurilor din colturi
  GLfloat Colors[] = {
    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,
  };

  glGenBuffers(1, &VboId);
  glBindBuffer(GL_ARRAY_BUFFER, VboId);
  glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);

  glGenVertexArrays(1, &VaoId);
  glBindVertexArray(VaoId);
  glEnableVertexAttribArray(0);
  glVertexAttribPointer(0, 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("shaderMatrice.vert", "shaderCuloare.frag");
  glUseProgram(ProgramId);
}
void DestroyShaders(void)
{
  glDeleteProgram(ProgramId);
}

void Initialize(void)
{
  // facem background-ul negru
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
  CreateVBO();
  CreateShaders();
  codColLocation = glGetUniformLocation(ProgramId, "codCuloare");
  myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
}
void RenderFunction(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
```

```
  resizeMatrix = glm::ortho(-windowWidth, windowWidth, -windowHeight, windowHeight);
  matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(positionX, positionY, 0.0));
  matrScale = glm::scale(glm::mat4(1.0f), glm::vec3(i, i, 0.0));
  matrRot = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0, 0.0, 1.0));

  myMatrix = resizeMatrix * matrTransl * matrScale * matrRot;

  if (codAction == 1)
  {
    if (i < 1.2) i = i + 0.0001;
  }
  else if (codAction == 2)
  {
    if (i > 0.8) i = i - 0.0001;
  }

  if (codAction == 3)
  {
    angle = angle - beta;
  }
  else
  {
    angle = 0.0;
  }

  glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
  glUniform1i(codColLocation, codCol);
  glDrawArrays(GL_POLYGON, 0, 4);

  glutSwapBuffers();
  glFlush();
}
void Cleanup(void)
{
  DestroyShaders();
  DestroyVBO();
}

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(windowWidth, windowHeight);
  glutCreateWindow("Bouncing Square");
  glewInit();
  Initialize();
  glutDisplayFunc(RenderFunction);
  glutMouseFunc(mouse);
  glutCloseFunc(Cleanup);
  glutMainLoop();
}
```