Grafică pe Calculator - Proiect 1

Simularea grafică a unui joc de biliard (9-ball) în 2D

Membrii echipei: Dilirici Mihai, Florea Mădălin-Alexandru, Nechita Maria-Ilinca

Grupa: 343

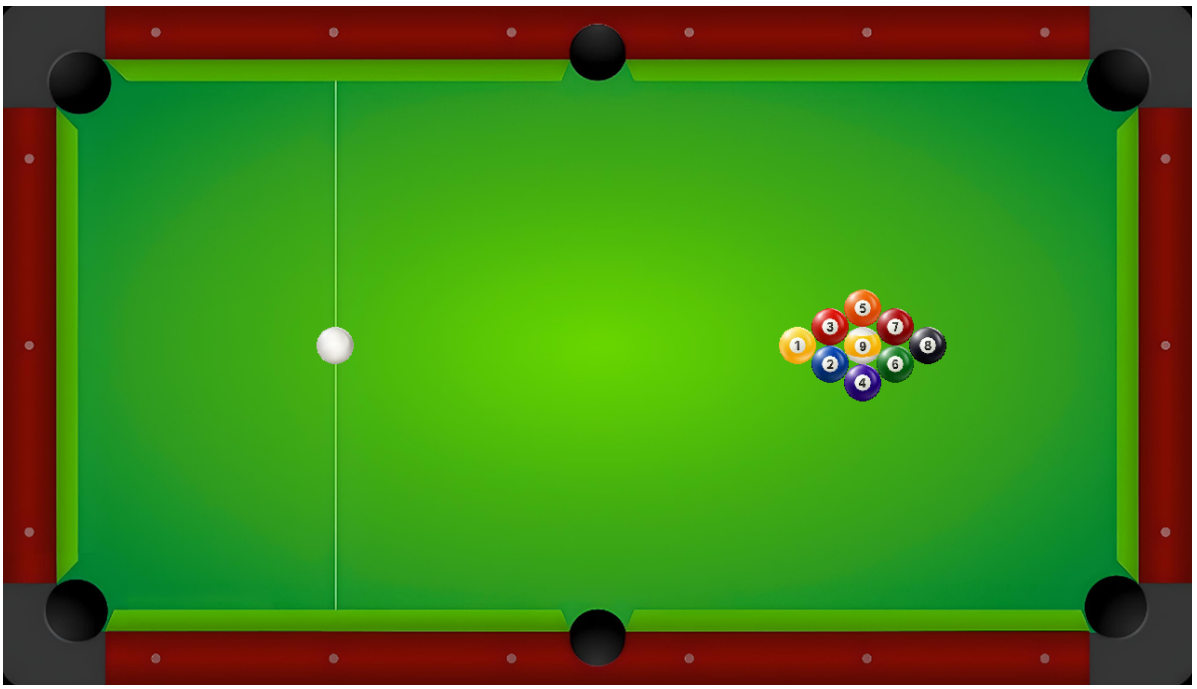
Cuprinsul documentației:

1. Conceptul proiectului

- Proiectul dezvoltat de noi simulează un joc de 9-Ball Pool în format 2D, în care jucătorul care a spart bilele este și câștigătorul meciului, reușind să își păstreze rândul după fiecare lovitură și să introducă în buzunar bila cu numărul 9.

- Regulile jocului:
  - Jucătorul trebuie să lovească întotdeauna bila cu cel mai mic număr aflată pe masă, dar nu este obligatoriu ca acea bilă să fie și cea introdusă în buzunar din lovitura respectivă.
  - Pentru ca un jucător să își păstreze rândul și să continue să lovească, trebuie să introducă cel puțin o bilă în buzunar.
  - Scopul principal al jocului este de a împinge bila cu numărul 9 într-un buzunar. Având în vedere regulile anterioare, pentru a câștiga jocul mai rapid, există posibilitatea de a introduce bila cu numărul 9 și înaintea altor bile cu număr mai mic (ceea ce am simulat și noi în proiectul nostru), dar cu condiția ca bila albă să lovească prima oară bila cu cel mai mic număr de pe masă.

- Aranjarea bilelor: la începutul jocului, bilele sunt aranjate într-un romb. Bila cu numărul 1 trebuie plasată în stânga rombului (pentru a fi cea mai apropiată bilă de cea albă), iar bila cu numărul 9 în centrul rombului:



2. Transformările utilizate

- Pentru a reda în mod cât mai realist coliziunile dintre bile (bila albă și cea pe care vrem să o lovim), ne-am folosit de un vector care reține la fiecare mutare distanța parcursă de bilele asupra cărora s-a efectuat o mutare, dar și un vector în care se află pozițiile bilelor (în acesta sunt reținute coordonatele x și y ale unei bile și este actualizat după fiecare mutare făcută).

- Mișcarea efectivă a bilelor se realizează prin operații de translatare efectuate asupra lor la fiecare pas. Spre exemplu, translația de mai jos corespunzând mișcării bilei albe are valorile *distances[0][0]* și *distances[0][1]*, pentru a specifica cât de mult să fie deplasată bila pe axele x și y.

  translationMatrices[0]=glm::translate(glm::mat4(1.0f),glm::vec3(distances[0][0], distances[0][1], 0.0));

- Modificarea valorilor din vectorul *distances* se va produce în funcțiile specifice fiecărui turn. De asemenea, tot în cadrul acestor funcții se produce mișcarea bilelor cu anumite unități pe fiecare axă în parte până se ajunge la coordonatele finale pentru mutarea respectivă.

  În secvența de cod de mai jos verificăm dacă bila albă a lovit bila galbenă (bila cu numărul 1), iar în caz contrar, continuăm să o mișcăm:

```
// Testam dacă bila alba a lovit bila galbena
if (!turn1_ball0_hit) {
        if (distances[0][0] >= ballsPositions[1][0] - ballsPositions[0][0] - 2 *
radius)
                turn1_ball0_hit = true;

        // Daca bila alba nu a lovit bila galbena, continuam să o miscăm
        distances[0][0] += 1.0;
}
```

Dacă bila albă a atins bila galbenă, toate bilele vor fi mutate în locuri diferite pe masă. De exemplu, după ce s-a realizat coliziunea bilei albe cu cea galbenă, aceasta ar trebui să se deplaseze înapoi în partea stângă a mesei:

```
// Mutam bila alba in partea stanga a mesei
if (distances[0][0] >= -105.0) // distanta fata de pozitia initiala a bilei albe
        distances[0][0] -= 0.5;
else
        turn1_ball0_destination = true;
```

Acesta este un exemplu de mutare a altei bile în urma ciocnirii dintre bila albă și cea galbenă, mai precis mutarea bilei albastre puțin spre stânga și mai jos:

```
// Mutam bila albastra putin mai jos si mai la stanga
if (distances[2][0] >= -75.0) {
        distances[2][0] -= 0.05;
        distances[2][1] -= 0.15;
}
else
        turn1_ball2_destination = true;
```

- În plus, pentru a putea trece la pasul următor, va trebui să ne asigurăm că toate bilele au fost mutate în locul dorit. Din acest motiv, utilizăm variabile de tip bool pentru fiecare deplasare în parte.

- După ce toate bilele au fost mutate corespunzător, actualizăm poziția fiecărei bile în matricea *ballsPositions*, resetăm matricea *distances* pentru distanțele de translație și marcăm turn-ul ca fiind finalizat:

```
// Verificam daca toate bilele au ajuns in locul corespunzator
if (turn2_ball0_destination && turn1_ball1_destination) {
        // Actualizam pozitia bilei albe
        ballsPositions[0][0] += distances[0][0];
        ballsPositions[0][1] += distances[0][1];

        // Resetam distantele de translatie ale bilei albe
        distances[0][0] = 0.0;
        distances[0][1] = 0.0;

        // Marcam turn-ul ca fiind finalizat
        turn2_finished = true;

        // Oprim animatia pana la urmatorul click
        glutIdleFunc(NULL);
}
```

- O altă transformare este dată de matricea resizeMatrix, folosită pentru proiecția ortogonală, prin care se redimensionează scena astfel încât coordonatele vârfurilor să se afle în intervalul [-640, 640] pe axa Ox și [-365, 365] pe axa Oy.

- Acțiunile efectuate pentru lovirea bilelor sunt declanșate prin intermediul click-ului stânga. Fiecare click face ca jocul să treacă la turn-ul următor, iar după finalizarea mișcării bilelor, imaginea rămâne fixată până la apăsarea următorului click. Pentru evitarea unui bug care ar putea apărea dacă un click este apăsat înainte de finalizarea animației în desfășurare, am introdus condiția ca un click să fie luat în considerare doar după ce deplasările sunt complete. După finalizarea ultimului turn, pe ecran va rămâne imaginea cu poziția finală a bilelor, peste care va apărea o textură conținând mesajul „Game Over!”, iar click-ul stânga nu va mai influența cu nimic imaginea. Pentru a ilustra cât mai bine cursul jocului, am realizat un demo care poate fi vizionat aici.

```
int clickCounter = 0;                                                    //
Numarul de click-uri apasate pe ecran
//  Functia folosita pentru a declansa fiecare lovire a bilei albe, folosind click-ul
stanga al mouse-ului
void UseMouse(int button, int state, int x, int y) {
        if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
                ++clickCounter;
```

```
            switch (clickCounter) {
            case 1:
                glutIdleFunc(Turn1);
                break;
            case 2:
                //  Daca se apasa click-ul stanga inainte de finalizarea turn-ului 1, se decrementeaza numarul de click-uri
                if (!turn1_finished)
                    --clickCounter;
                else
                    glutIdleFunc(Turn2);
                break;
            case 3:
                //  Daca se apasa click-ul stanga inainte de finalizarea turn-ului 2, se decrementeaza numarul de click-uri
                if (!turn2_finished)
                    --clickCounter;
                else
                    glutIdleFunc(Turn3);
                break;
            case 4:
                //  Daca se apasa click-ul stanga inainte de finalizarea turn-ului 3, se decrementeaza numarul de click-uri
                if (!turn3_finished)
                    --clickCounter;
                else
                    glutIdleFunc(Turn4);
                break;
            }
        }
}
```

3. Elemente de originalitate

- O primă notă originală este adusă prin aplicarea texturării atât mesei de biliard, cât și fiecărei bile. Pentru a realiza acest lucru, se calculează coordonatele x și y ale fiecărui punct al cercului prin intermediul formulei pentru coordonatele unui punct de pe un cerc:

  x = r * cos(θ), y = r * sin(θ), unde r este raza cercului, iar θ este unghiul.

  De asemenea, coordonatele de texturare sunt calculate și utilizate pentru maparea texturilor pe cerc, după formula:

  x = 0.5 * cos(θ), y = 0.5 * sin(θ), unde θ este unghiul.

```
//  Se creeaza coordonatele bilelor de biliard care n-au fost bagate in buzunar
for (int i = 0; i < ballsNumber; ++i) {
        if (!pottedBalls[i])
        for (int j = 0; j < circleVertices; ++j) {
                float angle = 2 * PI * j / circleVertices;
                Vertices[(8 + i * circleVertices + j) * 9]     = radius * cos(angle) +
ballsPositions[i][0];    // coordonata x
                Vertices[(8 + i * circleVertices + j) * 9 + 1] = radius * sin(angle) +
ballsPositions[i][1];    // coordonata y

                Vertices[(8 + i * circleVertices + j) * 9 + 7] = 0.5f * (1.0f +
cos(angle));                // coordonata de texturare x
                Vertices[(8 + i * circleVertices + j) * 9 + 8] = 0.5f * (1.0f +
sin(angle));                // coordonata de texturare y
        }
}
```

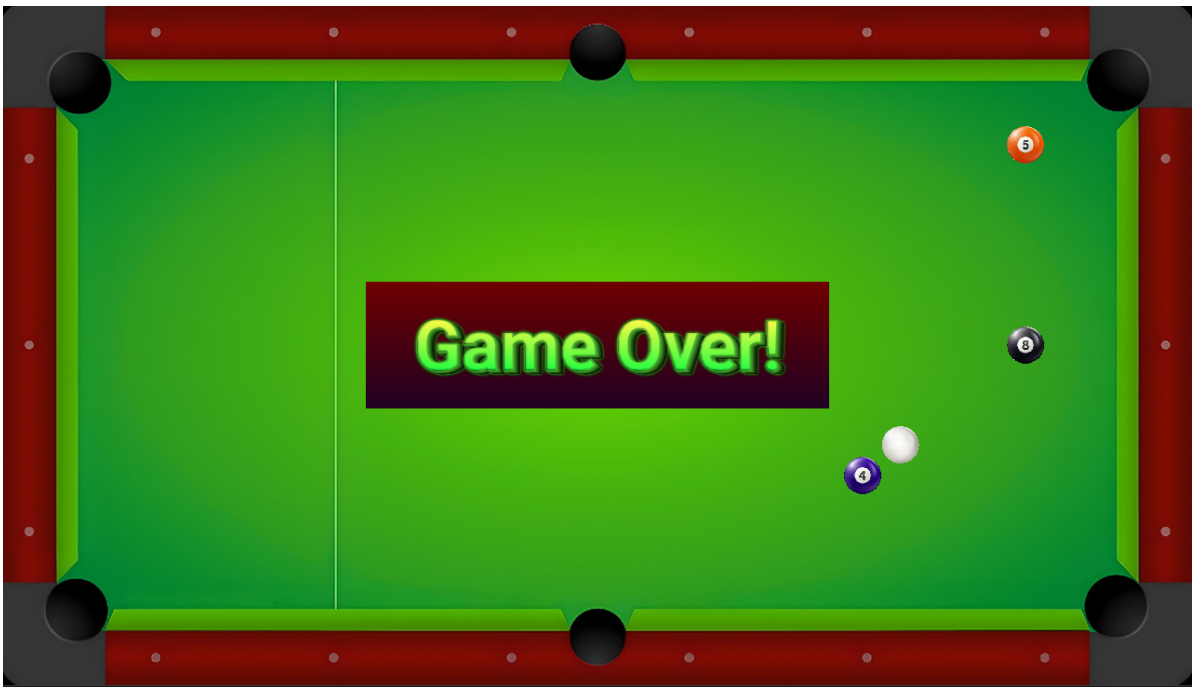
- Deoarece în OpenGL nu există opțiunea de a desena un cerc folosind o instrucțiune predefinită precum *GL_POLYGON*, am creat pentru fiecare bilă un număr de 25 de vârfuri și am decupat texturile pentru a se potrivi în interiorul poligoanelor respective.

- Mai mult decât atât, noi am gândit și calculat traiectoria fiecărei bile în parte la fiecare pas, respectând regulile de bază ale jocului și componenta fizică a mișcării bilelor. Astfel, viteza deplasării unei bile după ce a fost lovită a corespuns tipului de lovitură (bila albă avea viteza de deplasare cea mai mare, aceasta scăzând după ce a lovit altă bilă din cauza impactului dintre acestea).

4. Modificări aduse în plus după discuțiile purtate la laborator

- Am rezolvat un bug care apărea în cazul în care click-ul stânga era apăsat când bilele erau deja în mișcare (cu alte cuvinte, cât timp turn-ul precedent nu a fost finalizat). Acum, următorul turn este declanșat doar dacă click-ul stânga este apăsat după finalizarea turn-ului precedent.

- Am reușit să adăugăm o textură care să conțină mesajul „Game Over!” și să sugereze sfârșitul animației. Aceasta va apărea pe ecran după finalizarea ultimului turn:



5. Contribuții personale

- Realizarea proiectului s-a realizat în cadrul mai multor întâlniri online la care au participat toți membri echipei, procesul de proiectare și dezvoltare fiind unul colectiv.

- Au existat, de asemenea, părți ale proiectului în care anumite persoane din echipă au avut o contribuție mai mare. De exemplu, Mihai s-a concentrat mai mult pe aplicarea texturilor, Mădălin a implementat funcțiile de *turn*, iar Ilinca a lucrat la formule și desenarea traiectoriei bilelor.

6. Coduri sursă

- **Biliard_Shader.frag**

```
// ==================================================
// |          Grafica pe calculator          |
// ==================================================
// |        PROIECT 1 - Joc de Biliard        |
// ==================================================
//
//  Shaderul de fragment / Fragment shader - afecteaza culoarea pixelilor;

#version 330 core
```

```
//		Variabile de intrare (dinspre Shader.vert);
in vec4 ex_Color;
in vec2 tex_Coord;

//		Variabile de iesire	(spre programul principal);
out vec4 out_Color;		//		Culoarea actualizata;

// Variabile uniforme
uniform sampler2D myTexture;
uniform int drawCode;

void main(void) {
	out_Color = texture(myTexture, tex_Coord);
}
```

- **Biliard_Shader.vert**

```
// ==================================================
// |		Grafica pe calculator		|
// ==================================================
// |		PROIECT 1 - Joc de Biliard		|
// ==================================================
//
//  Shaderul de varfuri / Vertex shader - afecteaza geometria scenei;

#version 330 core

//  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;
```

```
void main(void) {
   gl_Position = myMatrix * in_Position;
   ex_Color = in_Color;
   tex_Coord = vec2(texCoord.x, 1 - texCoord.y);
}
```

- **Biliard.cpp**

```
// ================================================
//            Grafica pe calculator            |
// ================================================
//           PROIECT 1 - Joc de Biliard         |
// ================================================

//  Biblioteci
#include <string>                                   //        Biblioteca pentru lucrul cu siruri de caractere;
#include <windows.h>                                //        Utilizarea functiilor de sistem Windows (crearea de ferestre, manipularea fisierelor si directoarelor);
#include <stdlib.h>                                 //  Biblioteci necesare pentru citirea shaderelor;
#include <stdio.h>
#include <GL/glew.h>                                //  Definește prototipurile functiilor OpenGL si constantele necesare pentru programarea OpenGL moderna;
#include <GL/freeglut.h>                            //        Include functii pentru:
                                                    //                -
gestionarea ferestrelor si evenimentelor de tastatura si mouse,
                                                    //      - desenarea
de primitive grafice precum dreptunghiuri, cercuri sau linii,
                                                    //      - crearea de
meniuri si submeniuri;
#include "loadShaders.h"                            //        Fisierul care face legatura intre program si shadere;
#include "SOIL.h"                                   //            Biblioteca pentru texturare;
#include "glm/glm.hpp"                              //            Bibloteci utilizate pentru transformari grafice;
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtx/transform.hpp"
#include "glm/gtc/type_ptr.hpp"
```

```
const int ballsNumber = 10;
                        //  Numarul de bile;
const int circleVertices = 25;
                    //      Numarul de varfuri ce compun poligoanele ce reprezinta bilele;
const int radius = 20;
                                //          Raza bilelor;
const float PI = 3.14159265359;
                                //          Valoarea lui PI;

const GLfloat winWidth = 1280, winHeight = 730;
                    //          Dimensiunile ferestrei de afisare;
const float xMin = -640, xMax = 640.f, yMin = -365.0f, yMax = 365.0f;               //
Variabile pentru proiectia ortogonala;

GLuint VaoId, VboId, EboId, ProgramId, myMatrixLocation, textures[12];                  //
Identificatorii obiectelor de tip OpenGL;
glm::mat4 myMatrix, resizeMatrix, translationMatrices[10];
        //          Variabile catre matricile de transformare;

GLfloat ballsPositions[10][2];
                                //          Matrice pentru pozitiile in care se afla bilele;
GLfloat distances[10][2];
                                //  Matrice pentru distantele parcurse de bile prin translatii;
bool pottedBalls[10];
                                //          Vector pentru bilele cazute in buzunar;


//  Variabile pentru testele din Turn-ul 1
//
//  Variabile pentru testarea coliziunilor
bool turn1_ball0_hit, turn1_ball4_hit;
//  Variabile pentru testarea faptului ca bilele au ajuns in locul unde ne-am propus sa le
mutam
bool      turn1_ball0_destination,      turn1_ball1_destination,      turn1_ball2_destination,
turn1_ball3_destination, turn1_ball4_destination;
bool      turn1_ball5_destination,      turn1_ball6_destination,      turn1_ball7_destination,
turn1_ball8_destination, turn1_ball9_destination;
//  Variabila pentru testarea finalizarii turn-ului
bool turn1_finished;

void Turn1(void) {
        //  Testam daca bila alba a lovit bila galbena
        if (!turn1_ball0_hit) {
```

```
        if (distances[0][0] >= ballsPositions[1][0] - ballsPositions[0][0] - 2 * radius)
                turn1_ball0_hit = true;

        //  Daca bila alba nu a lovit bila galbena, continuam sa o miscam
        distances[0][0] += 1.0;
}
//  Daca bila alba a atins bila galbena, mutam toate bilele in locuri diferite pe masa
else {
        //  Mutam bila alba in partea stanga a mesei
        if (distances[0][0] >= -105.0) //  distanta fata de pozitia initiala a bilei albe
                distances[0][0] -= 0.5;
        else
                turn1_ball0_destination = true;

        //  Mutam bila galbena putin mai jos si mai la stanga
        if (distances[1][0] >= -350.0) {
                distances[1][0] -= 0.3;
                distances[1][1] -= 0.15;
        }
        else
                turn1_ball1_destination = true;

        //  Mutam bila albastra putin mai jos si mai la stanga
        if (distances[2][0] >= -75.0) {
                distances[2][0] -= 0.05;
                distances[2][1] -= 0.15;
        }
        else
                turn1_ball2_destination = true;

        //  Mutam bila rosie pe centrul mesei, mai sus
        if (distances[3][0] >= -235.0) {
                distances[3][0] -= 0.15;
                distances[3][1] += 0.1;
        }
        else
                turn1_ball3_destination = true;

        //  Mutam bila mov mai jos (cu lovitura de manta)
        if (!turn1_ball4_hit) {
                //  Testam daca bila mov a lovit manta
                if (distances[4][1] <= -223.0)
                        turn1_ball4_hit = true;
```

```
		//  Daca nu a lovit manta, continuam sa o miscam
		distances[4][1] -= 0.2;
}
else {
		if (distances[4][1] <= -100.0)
			distances[4][1] += 0.1;
		else
			turn1_ball4_destination = true;
}

//  Mutam bila portocalie aproape de buzunarul din dreapta sus
if (distances[5][0] <= 175.0) {
		distances[5][0] += 0.1;
		distances[5][1] += 0.1;
}
else
		turn1_ball5_destination = true;

//  Mutam bila verde in buzunarul din stanga jos
if (distances[6][0] <= 250.0) {
		distances[6][0] += 0.2;
		distances[6][1] -= 0.2;
}
else {
		turn1_ball6_destination = true;
		pottedBalls[6] = true;
}

//  Mutam bila maro in buzunarul din dreapta sus
if (distances[7][0] <= 250.0) {
		distances[7][0] += 0.3;
		distances[7][1] += 0.3;
}
else {
		turn1_ball7_destination = true;
		pottedBalls[7] = true;
}

//  Mutam bila neagra putin la dreapta
if (distances[8][0] <= 105.0)
		distances[8][0] += 0.1;
else
		turn1_ball8_destination = true;
```

```
		//  Mutam bila cu numarul 9 foarte putin mai jos si mai la dreapta
		if (distances[9][0] <= 50.0) {
			distances[9][0] += 0.05;
			distances[9][1] -= 0.05;
		}
		else
			turn1_ball9_destination = true;
	}

	//  Verificam daca toate bilele au ajuns in locul corespunzator
	if (turn1_ball0_destination && turn1_ball1_destination && turn1_ball2_destination && turn1_ball3_destination && turn1_ball4_destination && turn1_ball5_destination && turn1_ball6_destination && turn1_ball7_destination && turn1_ball8_destination) {
		//  Actualizam pozitia fiecarei bile
		for (int i = 0; i < ballsNumber; ++i)
			for (int j = 0; j < 2; ++j)
				ballsPositions[i][j] += distances[i][j];

		//  Resetam vectorul pentru distantele de translatie
		for (int i = 0; i < ballsNumber; ++i)
			for (int j = 0; j < 2; ++j)
				distances[i][j] = 0.0;

		//  Marcam turn-ul ca fiind finalizat
		turn1_finished = true;

		//  Oprim animatia pana la urmatorul click
		glutIdleFunc(NULL);
	}
	glutPostRedisplay();
}


//	Variabile pentru testele din Turn-ul 2
//
//  Variabile pentru testarea coliziunilor
bool turn2_ball0_hit;
//  Variabile pentru testarea faptului ca bilele au ajuns in locul unde ne-am propus sa le mutam
bool turn2_ball0_destination, turn2_ball1_destination;
//  Variabila pentru testarea finalizarii turn-ului
bool turn2_finished;

void Turn2(void) {
```

```
	//  Testam daca bila alba a lovit bila galbena
	if (!turn2_ball0_hit) {
		if (distances[0][0] >= ballsPositions[1][0] - ballsPositions[0][0] - 2 * radius +
10.0)
			turn2_ball0_hit = true;

		//  Daca bila alba nu a lovit bila galbena, continuam sa o miscam
		distances[0][0] += 0.3;
		distances[0][1] -= 0.2;
	}
	//  Daca bila alba a atins bila galbena, mutam bilele corespunzatoare
	else {
		//  Mutam bila alba putin mai sus si mai la dreapta
		if (distances[0][0] <= 340.0) {
			distances[0][0] += 0.07;
			distances[0][1] -= 0.01;
		}
		else
			turn2_ball0_destination = true;

		//  Mutam bila galbena in buzunarul de jos din mijloc
		if (distances[1][0] <= 145.0) {
			distances[1][0] += 0.1;
			distances[1][1] -= 0.085;
		}
		else {
			turn1_ball1_destination = true;
			pottedBalls[1] = true;
		}
	}

	//  Verificam daca toate bilele au ajuns in locul corespunzator
	if (turn2_ball0_destination && turn1_ball1_destination) {
		//  Actualizam pozitia bilei albe
		ballsPositions[0][0] += distances[0][0];
		ballsPositions[0][1] += distances[0][1];

		//  Resetam distantele de translatie ale bilei albe
		distances[0][0] = 0.0;
		distances[0][1] = 0.0;

		//  Marcam turn-ul ca fiind finalizat
		turn2_finished = true;
```

```
			//  Oprim animatia pana la urmatorul click
			glutIdleFunc(NULL);
		}

		glutPostRedisplay();
}


//		Variabile pentru testele din Turn-ul 3
//
//  Variabile pentru testarea coliziunilor
bool turn3_ball0_hit1, turn3_ball0_hit2;
//  Variabile pentru testarea faptului ca bilele au ajuns in locul unde ne-am propus sa le mutam
bool turn3_ball0_destination, turn3_ball2_destination;
//  Variabila pentru testarea finalizarii turn-ului
bool turn3_finished;

void Turn3(void) {
		//  Testam daca bila alba a lovit bila albastra
		if (!turn3_ball0_hit1) {
			if (distances[0][0] >= ballsPositions[2][0] - ballsPositions[0][0] - 2 * radius)
				turn3_ball0_hit1 = true;

			//  Daca bila alba nu a lovit bila albastra, continuam sa o miscam
			distances[0][0] += 0.2;
			distances[0][1] -= 0.085;
		}
		//  Daca bila alba a atins bila albastra, mutam bilele corespunzatoare
		else {
			//  Mutam bila alba mai sus, la dreapta (dar dupa ce a lovit manta)
			//
			//  Testam daca bila alba a lovit manta
			if (!turn3_ball0_hit2) {
				if (distances[0][1] <= -98.0)
					turn3_ball0_hit2 = true;

				//  Daca bila alba nu a lovit manta, continuam sa o miscam
				distances[0][0] += 0.03;
				distances[0][1] -= 0.1;
			}
			//  Daca bila alba a lovit manta, o deplasam mai sus, la dreapta
			else {
				if (distances[0][1] <= -30.0) {
```

```
                        distances[0][0] += 0.035;
                        distances[0][1] += 0.07;
                }
                else
                        turn3_ball0_destination = true;
        }

        //  Mutam bila albastra in buzunarul din dreapta jos
        if (distances[2][0] <= 380.0) {
                distances[2][0] += 0.2;
                distances[2][1] -= 0.009;
        }
        else {
                turn3_ball2_destination = true;
                pottedBalls[2] = true;
        }
    }

    //  Verificam daca toate bilele au ajuns in locul corespunzator
    if (turn3_ball0_destination && turn3_ball2_destination) {
        //  Actualizam pozitia bilei albe
        ballsPositions[0][0] += distances[0][0];
        ballsPositions[0][1] += distances[0][1];

        //  Resetam distantele de translatie ale bilei albe
        distances[0][0] = 0.0;
        distances[0][1] = 0.0;

        //  Marcam turn-ul ca fiind finalizat
        turn3_finished = true;

        //  Oprim animatia pana la urmatorul click
        glutIdleFunc(NULL);
    }

    glutPostRedisplay();
}


//      Variabile pentru testele din Turn-ul 4
//
//  Variabile pentru testarea coliziunilor
bool turn4_ball0_hit1, turn4_ball0_hit2, turn4_ball0_hit3;
```

```
//  Variabile pentru testarea faptului ca bilele au ajuns in locul unde ne-am propus sa le mutam
bool turn4_ball0_destination, turn4_ball3_destination, turn4_ball9_destination;
//  Variabila pentru testarea finalizarii turn-ului
bool turn4_finished;

void Turn4(void) {
	//  Testam daca bila alba a lovit bila rosie
	if (!turn4_ball0_hit1) {
		if (distances[0][1] >= ballsPositions[3][1] - ballsPositions[0][1] - 11.0)
			turn4_ball0_hit1 = true;

		//  Daca bila alba nu a lovit bila rosie, continuam sa o miscam
		distances[0][0] -= 0.17;
		distances[0][1] += 0.5;
	}
	//  Daca bila alba a atins bila rosie, mutam bilele corespunzatoare
	else {
		//  Mutam bila rosie in buzunarul din stanga sus
		if (distances[3][0] >= -565.0) {
			distances[3][0] -= 0.4;
			distances[3][1] += 0.06;
		}
		else {
			turn4_ball3_destination = true;
			pottedBalls[3] = true;
		}

		//  Lovim bila alba de manta, deplasand-o in dreapta sus
		//
		//  Testam daca bila alba a lovit manta
		if (!turn4_ball0_hit2) {
			if (distances[0][1] >= 457)
				turn4_ball0_hit2 = true;

			//  Daca bila alba inca nu a lovit manta, continuam sa o miscam
			distances[0][0] += 0.3;
			distances[0][1] += 0.3;
		}
		//  Daca bila alba a lovit manta, o deplasam in dreapta jos catre bila galbena (9 ball)
		else {
			//  Testam daca bila alba a lovit bila galbena (9 ball)
			if (!turn4_ball0_hit3) {
```

```
                        if (distances[0][0] >= ballsPositions[9][0] - ballsPositions[0][0] - 2 * radius + 13.0)
                            turn4_ball0_hit3 = true;

                        // Daca bila alba nu a lovit bila galbena (9 ball), continuam sa o miscam
                        distances[0][0] += 0.13;
                        distances[0][1] -= 0.23;
                    }
                    // Daca bila alba a lovit bila galbena (9 ball), continuam sa miscam bilele corespunzatoare
                    else {
                        // Mutam bila galbena (9 ball) in buzunarul din dreapta jos
                        if (distances[9][0] <= 230.0) {
                            distances[9][0] += 0.1;
                            distances[9][1] -= 0.1;
                        }
                        else {
                            turn4_ball9_destination = true;
                            pottedBalls[9] = true;
                        }

                        // Mutam bila alba in jos si putin la dreapta
                        if (distances[0][0] <= 150.0) {
                            distances[0][0] += 0.01;
                            distances[0][1] -= 0.05;
                        }
                        else
                            turn4_ball0_destination = true;
                    }
                }
            }

            // Verificam daca toate bilele au ajuns in locul corespunzator
            if (turn4_ball0_destination && turn4_ball3_destination && turn4_ball9_destination) {
                // Marcam turn-ul ca fiind finalizat
                turn4_finished = true;

                // Oprim animatia
                glutIdleFunc(NULL);
            }

            glutPostRedisplay();
```

```
}


int clickCounter = 0;                                                                    //
Numarul de click-uri apasate pe ecran
//  Functia folosita pentru a declansa fiecare lovire a bilei albe, folosind click-ul stanga al
mouse-ului
void UseMouse(int button, int state, int x, int y) {
        if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
                ++clickCounter;

                switch (clickCounter) {
                case 1:
                        glutIdleFunc(Turn1);
                        break;
                case 2:
                        //  Daca se apasa click-ul stanga inainte de finalizarea turn-ului 1, se
decrementeaza numarul de click-uri
                        if (!turn1_finished)
                                --clickCounter;
                        else
                                glutIdleFunc(Turn2);
                        break;
                case 3:
                        //  Daca se apasa click-ul stanga inainte de finalizarea turn-ului 2, se
decrementeaza numarul de click-uri
                        if (!turn2_finished)
                                --clickCounter;
                        else
                                glutIdleFunc(Turn3);
                        break;
                case 4:
                        //  Daca se apasa click-ul stanga inainte de finalizarea turn-ului 3, se
decrementeaza numarul de click-uri
                        if (!turn3_finished)
                                --clickCounter;
                        else
                                glutIdleFunc(Turn4);
                        break;
                }
        }
}
```

```
//	Functia de incarcare a texturilor in program;
void LoadTexture(const char* photoPath, int index) {
	glGenTextures(1, &textures[index]);
	glBindTexture(GL_TEXTURE_2D, textures[index]);

	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_RGB);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
	glGenerateMipmap(GL_TEXTURE_2D);

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


//  Crearea si compilarea obiectelor de tip shader;
void CreateShaders(void) {
	ProgramId = LoadShaders("Biliard_Shader.vert", "Biliard_Shader.frag");
	glUseProgram(ProgramId);
}


//  Se initializeaza un Vertex Buffer Object (VBO) pentru tranferul datelor spre memoria placii grafice (spre shadere);
//  In acesta se stocheaza date despre varfuri (coordonate, culori, indici, texturare etc.);
void CreateVBO(void) {
	//  Se initializeaza un vector in care se vor retine datele pentru varfuri;
	//  Acesta va contine - 4 varfuri pentru masa de biliard
	//                    - 4 varfuri pentru mesajul de Game Over
	//                    - ballsNumber * circleVertices varfuri pentru bilele de biliard
	//  Fiecare varf are 9 valori asociate: 4 pentru pozitie, 3 pentru culoare, 2 pentru texturare (in aceasta ordine);
	GLfloat Vertices[(ballsNumber * circleVertices + 8) * 9];
```

```
	//  MASA DE BILIARD

	//  Se adauga manual coordonatele varfurilor mesei de biliard
	//    coordonata x   |    coordonata y    |        coordonate de texturare
	Vertices[0]  = -640.0f; Vertices[1]  = -365.0f; Vertices[7]  = 0.0f; Vertices[8]  = 0.0f;
	Vertices[9]   =   640.0f;  Vertices[10] = -365.0f;  Vertices[16] = 1.0f;  Vertices[17] =
0.0f;
	Vertices[18] =   640.0f;  Vertices[19] =   365.0f;  Vertices[25] = 1.0f;  Vertices[26] =
1.0f;
	Vertices[27] = -640.0f;  Vertices[28] =   365.0f;  Vertices[34] = 0.0f;  Vertices[35] =
1.0f;


	//  MESAJUL DE GAME OVER

	//  Se adauga manual coordonatele varfurilor mesajului de Game Over
	//    coordonata x   |    coordonata y    |        coordonate de texturare
	Vertices[36] = -249.0f;  Vertices[37] =   -68.0f;  Vertices[43] = 0.0f;  Vertices[44] =
0.0f;
	Vertices[45] =   249.0f;  Vertices[46] =   -68.0f;  Vertices[52] = 1.0f;  Vertices[53] =
0.0f;
	Vertices[54] =   249.0f;  Vertices[55] =    68.0f;  Vertices[61] = 1.0f;  Vertices[62] =
1.0f;
	Vertices[63] = -249.0f;  Vertices[64] =    68.0f;  Vertices[70] = 0.0f;  Vertices[71] =
1.0f;


	//  BILELE DE BILIARD

	//  Se creeaza coordonatele bilelor de biliard care n-au fost bagate in buzunar
	for (int i = 0; i < ballsNumber; ++i) {
		if (!pottedBalls[i])
		for (int j = 0; j < circleVertices; ++j) {
			float angle = 2 * PI * j / circleVertices;
			Vertices[(8 + i * circleVertices + j) * 9]      = radius * cos(angle) +
ballsPositions[i][0];   // coordonata x
			Vertices[(8 + i * circleVertices + j) * 9 + 1] = radius * sin(angle) +
ballsPositions[i][1];   // coordonata y

			Vertices[(8 + i * circleVertices + j) * 9 + 7] = 0.5f * (1.0f +
cos(angle));              // coordonata de texturare x
```

```
                    Vertices[(8 + i * circleVertices + j) * 9 + 8] = 0.5f * (1.0f + sin(angle));                // coordonata de texturare y
                }
        }


        //  Se modifica aceste coordonate pentru toate varfurile din scena, fiind identice
        for (int i = 0; i < ballsNumber * circleVertices + 8; i++) {
                Vertices[i * 9 + 2] = 0.0f;
                 // coordonata z
                Vertices[i * 9 + 3] = 1.0f;
            // coordonata w

                Vertices[i * 9 + 4] = 0.0f;
                 // valoarea pentru culoarea rosie
                Vertices[i * 9 + 5] = 0.0f;
                 // valoarea pentru culoarea verde
                Vertices[i * 9 + 6] = 0.0f;
                 // valoarea pentru culoarea albastra
        }


        //      Indicii care determina ordinea de parcurgere a varfurilor;
        static const GLuint Indices[] = {
                0, 1, 2, 3,  //  Indicii pentru masa de biliard
                4, 5, 6, 7,  //  Indicii pentru mesajul de Game Over
        };


        //  Transmiterea datelor prin buffere;

        //  Se creeaza / se leaga un VAO (Vertex Array Object) - util cand se utilizeaza mai multe VBO;
        glGenVertexArrays(1, &VaoId);                                   //  Generarea VAO si indexarea acestuia catre variabila VaoId;
        glBindVertexArray(VaoId);

        //  Se creeaza un buffer pentru VARFURI - COORDONATE, CULORI si TEXTURARE;
        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);

	//	Se creeaza un buffer pentru INDICI;
	glGenBuffers(1, &EboId);
                                        //  Generarea bufferului si indexarea acestuia catre variabila EboId;
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId);
                                    //  Setarea tipului de buffer - atributele varfurilor;
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices, GL_STATIC_DRAW);

	//	Se activeaza lucrul cu atribute;
	glEnableVertexAttribArray(0);                   //  Se asociaza atributul (0 = coordonate) pentru shader;
	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 9 * sizeof(GLfloat), (GLvoid*)0);
	glEnableVertexAttribArray(1);                   //  Se asociaza atributul (1 = culoare) pentru shader;
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 9 * sizeof(GLfloat), (GLvoid*)(4 * sizeof(GLfloat)));
	glEnableVertexAttribArray(2);                   //  Se asociaza atributul (2 = texturare) pentru shader;
	glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 9 * sizeof(GLfloat), (GLvoid*)(7 * sizeof(GLfloat)));
}


//  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, texturare etc.);
	glDisableVertexAttribArray(2);
	glDisableVertexAttribArray(1);
	glDisableVertexAttribArray(0);
```

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

	//  Eliberarea 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 vizualizare;
void Initialize(void) {
	glClearColor(1.0f, 1.0f, 1.0f, 1.0f);                    //  Culoarea de fond a ecranului;
	CreateShaders();
	//  Initilizarea shaderelor;

	//	Instantierea variabilei uniforme pentru a "comunica" cu shaderele;
	myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");

	resizeMatrix = glm::ortho(xMin, xMax, yMin, yMax);          //  Matricea pentru proiectia ortogonala;


	//  Initializarea vectorului cu pozitiile bilelor
	//	coordonata x	|	coordonata y
	ballsPositions[0][0] = -282.0f; ballsPositions[0][1] =   0.0f;
	ballsPositions[1][0] =  215.0f; ballsPositions[1][1] =   0.0f;
	ballsPositions[2][0] =  250.0f; ballsPositions[2][1] = -20.0f;
	ballsPositions[3][0] =  250.0f; ballsPositions[3][1] =  20.0f;
	ballsPositions[4][0] =  285.0f; ballsPositions[4][1] = -40.0f;
	ballsPositions[5][0] =  285.0f; ballsPositions[5][1] =  40.0f;
	ballsPositions[6][0] =  320.0f; ballsPositions[6][1] = -20.0f;
	ballsPositions[7][0] =  320.0f; ballsPositions[7][1] =  20.0f;
	ballsPositions[8][0] =  355.0f; ballsPositions[8][1] =   0.0f;
	ballsPositions[9][0] =  285.0f; ballsPositions[9][1] =   0.0f;
```

```
	//  Incarcarea texturilor o singura data pentru o performanta mai buna a programului
	//  Vectorul textures va contine texturile dupa cum urmeaza:
	//  - de la pozitia 0 la pozitia 9 vor fi texturile pentru bilele de biliard
	//  - pe pozitia 10 va fi textura pentru masa de biliard
	//  - pe pozitia 11 va fi textura pentru mesajul de Game Over

	//  Incarcarea texturii pentru fiecare bila
	for (int i = 0; i < ballsNumber; ++i) {
		std::string textureFileName = "./textures/" + std::to_string(i) + "ball.png";
		LoadTexture(textureFileName.c_str(), i);
	}

	//  Incarcarea texturii pentru masa de biliard
	LoadTexture("./textures/Table.png", 10);

	//  Incarcarea texturii pentru mesajul de Game Over
	LoadTexture("./textures/GameOver.png", 11);
}


//  Functia de desenarea a graficii pe ecran;
void RenderFunction(void) {
	glClear(GL_COLOR_BUFFER_BIT);
	CreateVBO();

	//  Desenarea mesei de biliard
	glActiveTexture(GL_TEXTURE0);
	glBindTexture(GL_TEXTURE_2D, textures[10]);
	glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

	myMatrix = resizeMatrix;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_POLYGON, 4, GL_UNSIGNED_INT, (void*)(0));


	//  Desenarea bilelor de biliard
	for (int i = 0; i < ballsNumber; ++i) {
		//  Daca bila este cazuta in buzunar, nu se mai deseneaza
		if (!pottedBalls[i]) {
			//  Se creaza matricea de translatie pentru fiecare bila;
```

```
                translationMatrices[i] = glm::translate(glm::mat4(1.0f), glm::vec3(distances[i][0], distances[i][1], 0.0));

                // Se activeaza textura si se trimite la shader;
                glActiveTexture(GL_TEXTURE0);
                glBindTexture(GL_TEXTURE_2D, textures[i]);
                glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

                // Se trimite matricea de translatie la shader;
                myMatrix = resizeMatrix * translationMatrices[i];
                glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

                // Se deseneaza bila de biliard;
                glDrawArrays(GL_POLYGON, 8 + i * circleVertices, circleVertices);
            }
        }


        // Desenarea mesajului de Game Over
        if (turn4_finished) {
            glActiveTexture(GL_TEXTURE0);
            glBindTexture(GL_TEXTURE_2D, textures[11]);
            glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

            myMatrix = resizeMatrix;
            glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
            glDrawElements(GL_POLYGON, 4, GL_UNSIGNED_INT, (void*)(4 * sizeof(GLuint)));
        }

        glutSwapBuffers();
        glFlush();
}


int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);                    // Modul de afisare al ferestrei, se foloseste un singur buffer de afisare si culori RGB;
        glutInitWindowSize(winWidth, winHeight);                        // Dimensiunea ferestrei;
        glutInitWindowPosition(0, 0);                                   // Pozitia initiala a ferestrei;
```

```
	glutCreateWindow("Biliard");                                    //
	Creeaza fereastra de vizualizare, indicand numele acesteia;

	glewInit();

	Initialize();
	glutDisplayFunc(RenderFunction);                            //
Desenarea scenei in fereastra;
	glutMouseFunc(UseMouse);                                        //
	Activarea utilizarii mouseului;
	glutCloseFunc(Cleanup);
//  Eliberarea resurselor alocate de program;

	glutMainLoop();
	return 0;
}
```