Staicu Octavian-Florin – Grupa 331

Barbu Iulia Andreea – Grupa 331

Snowman

~ Documentație ~

Conceptul proiectului

Proiectul reprezintă un joc unde caracterul este un om de zăpadă care se poate plimba la stânga și la dreapta. Dacă depașește o margine a ecranului, se va teleporta la începutul părții opuse. Scopul lui este să prindă cât mai mulți fulgi, pentru a crește „nemăsurabil de mare”. Atunci când ratează un fulg, va fi penalizat și va descreste cu 4 unități. Scorul fulgilor adunati de el este afișat în bara ferestrei de joc, la fel și highscorul (care este ținut minte în fișierul local snowman_Highscore.txt) si viața. La prea mulți fulgi pierduți (când s-a topit complet omul de zăpadă), va apărea in bară mesajul “Game over” si ninsoarea se va opri.

Transformări incluse

Scalări:

- Am folosit matricea de scalare resizeMatrix pentru a discretiza intervalul ecranului de la [0, 800] x [0, 600] la [-1, 1] si a se putea realiza desenarea.

```
219     resizeMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(1.f / 400, 1.f / 300, 1.0));
220     myMatrix = resizeMatrix * matrDeplasare;
221
```

- Capul omului de zapada a fost obtinut prin scalarea (micsorarea) si translatarea in sus (cresterea coordonatei Oy) a “poligonului corp”:

```
glm::mat4 headScale = glm::scale(glm::mat4(1.0f), glm::vec3(2.f / 3, 2.f / 3, 1.0));
glm::mat4 headTransl1 = glm::translate(glm::mat4(1.0f), glm::vec3(-200.f, -100.f, 0.0));
glm::mat4 headTransl2 = glm::translate(glm::mat4(1.0f), glm::vec3(200.f, 100.f, 0.0));
glm::mat4 headUpTransl = glm::translate(glm::mat4(1.0f), glm::vec3(0.f, 137.5f, 0.0));

headAllTranslates = headUpTransl * headTransl2 * headScale * headTransl1;
```

Aplicarea in RenderFunction:

```
474     //// Snowman's body
475     codCol = 2; // 2 == white
476     codColLocation = glGetUniformLocation(ProgramId, "codCol");
477     glUniform1i(codColLocation, codCol);
478     glDrawArrays(GL_TRIANGLE_FAN, 16, 12);
479
480     // Snowman's head
481     myMatrix = myMatrix * headAllTranslates;
482     myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
483     glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
484     glDrawArrays(GL_TRIANGLE_FAN, 16, 12);
485
```

- Omului de zapada i se recalculează o matrice de scalare de fiecare dată când prinde sau pierde un fulg (dacă îl prinde atunci crește, daca il pierde atunci se micsoreaza).

Se calculeaza in variabila globala “grow” valoarea cu care sa fie scalat si apoi se calculeaza matricea growScale:

```
344 void growUpSnowman() {
345     grow += growFactor;
346     cout << "\ngrow_up_snowman: " << grow << "\n";
347     if (grow >= 2.0f)
348         grow = 2.0f;
349     growScale = glm::scale(glm::mat4(1.0f), glm::vec3(grow, grow, 1.0));
350 }
351
352
353 void shrinkSnowman() {
354     grow -= growFactor * 4;
355     if (grow < 0.0f) {
356         grow = 0.0f;
357     }
358     cout << "\nshrink_snowman: " << grow << "\n";
359
360     growScale = glm::scale(glm::mat4(1.0f), glm::vec3(grow, grow, 1.0));
361 }
362
```

Aplicarea in RenderFunction:

```
460     //// grow/shrink and move Snowman ( if needed)
461     moveTransl = glm::translate(glm::mat4(1.0f), glm::vec3(tx, 0.f, 0.0));
462     glm::mat4 growAux = growTransl2 * growScale * growTransl1;
463     myMatrix = myMatrix * moveTransl * growAux;
```

Rotații:

- Fulgii se rotesc in timpul caderii:

Calculare:

```
221     //rotate snowflake
222     snowflakeTranslRot = glm::translate(glm::mat4(1.0f), glm::vec3(-7.5, -510, 0.0));
223     snowflakeTranslRotInv = glm::translate(glm::mat4(1.0f), glm::vec3(7.5, 510, 0.0));
```

Aplicare:

```
408         // fulg cu centru de (7.5, 510)
409         glm::mat4 rotate_aux = snowflakeTranslRotInv * snowflakeRotate * snowflakeTranslRot; //put the bas
410         glm::mat4 rotateSnowflake = resizeMatrix * matrDeplasare * snowflakeTransl * rotate_aux;// rotate
411         myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
412         glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &rotateSnowflake[0][0]);
413         drawSnowflake();
```

Translatii:

- Ochiul drept al omului de zapada a fost obtinut prin translatarea ochiului stang:

```
eyeTransl = glm::translate(glm::mat4(1.0f), glm::vec3(50.f, 0.f, 0.0));
```

- Omul de zapada este translatat la stanga si la dreapta in functie de tastele pe care apasa jucatorul:

Este updatat un tx global care reprezinta distanta cu care omul de zapada o sa fie translatat pe axa Ox:

```
249 void processSpecialKeys(int key, int x, int y)
250 {
251     switch (key) {
252     case GLUT_KEY_LEFT:
253         // chiar daca ecranul meu e intre 0 - 800, eu am plasat omul de zapada sa stea initial undeva pe la poz 250
254         // si atunci, tx == 0 atunci cand omul de zapada e la pozitia 250
255         // deci d aia tx e decalat cam cu -250 fata de marginile reale ale ecranului
256         if (tx <= -250)
257             tx = 650;
258         else
259             tx -= 10.f;
260         break;
261
262     case GLUT_KEY_RIGHT:
263         if (tx >= 650)
264             tx = -250;
265         else
266             tx += 10.f;
267         break;
268     default:
269         break;
270     }
271 }
272
```

Apoi se calculeaza si se aplica matricea moveTransl:

```
460     //// grow and move Snowman ( if needed)
461     moveTransl = glm::translate(glm::mat4(1.0f), glm::vec3(tx, 0.f, 0.0));
462     glm::mat4 growAux = growTransl2 * growScale * growTransl1;
463     myMatrix = myMatrix * moveTransl * growAux;
```

- Fulgii care compun ninsoarea sunt obtinuti din fulgul initial translatat cu o valoare random pe axa Ox si redesenat.

Stocam valorile de translatie pentru fiecare fulg in vectorul de perechi snowflakes astfel:

```
274 void generateSnowflakes()
275 {
276     current_time = time(NULL);
277
278     if (current_time != last_time)
279     {
280         last_time = current_time;
281         int randX = rand() % 700 + 100;
282         cout << randX << " ";
283         snowflakes.push_back(make_pair(randX, 0));
284     }
285 }
```

Unde:

- **Snowflake[i].first** reprezintă valoarea cu care va fi translatat fulgul inițial pe axa Ox înainte de a fi redesenat
- **Snowflake[i].second** reprezintă valoarea cu care va fi translatat fulgul inițial pe axa Oy înainte de a fi redesenat

Iar apoi in interiorul for-ului care parcurge vectorul aplicăm translația si apelăm funcția de desenare a fulgului:

```
408     // fulg cu centru de (7.5, 510)
409     glm::mat4 rotate_aux = snowflakeTranslRotInv * snowflakeRotate * snowflakeTranslRot; //put th
410     glm::mat4 rorateSnowflake = resizeMatrix * matrDeplasare * snowflakeTransl * rotate_aux;   //
411     myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
412     glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &rorateSnowflake[0][0]);
413     drawSnowflake();
```

- Fiecare fulg este translatat cu –20 pe axa Oy o dată la fiecare secunda.

Este apelată din RenderFunction funcția drawSnowflakes. Parametrul funcției reprezintă cu cat o sa fie translatat fulgul **in jos** pe axa Oy. Dacă nu a trecut o secundă, va fi apelată cu valoarea 0.

```
450     current_time2 = time(NULL);
451
452     if (current_time2 != last_time2)
453     {
454         last_time2 = current_time2;
455         snowflakeRotate = glm::rotate_slow(glm::mat4(1.0f), (++alpha) * PI / 8, glm::vec3(0.0, 0.0, 1.0));
456         drawSnowflakes(20);
457     }
458     else
459         drawSnowflakes(0);
460
```

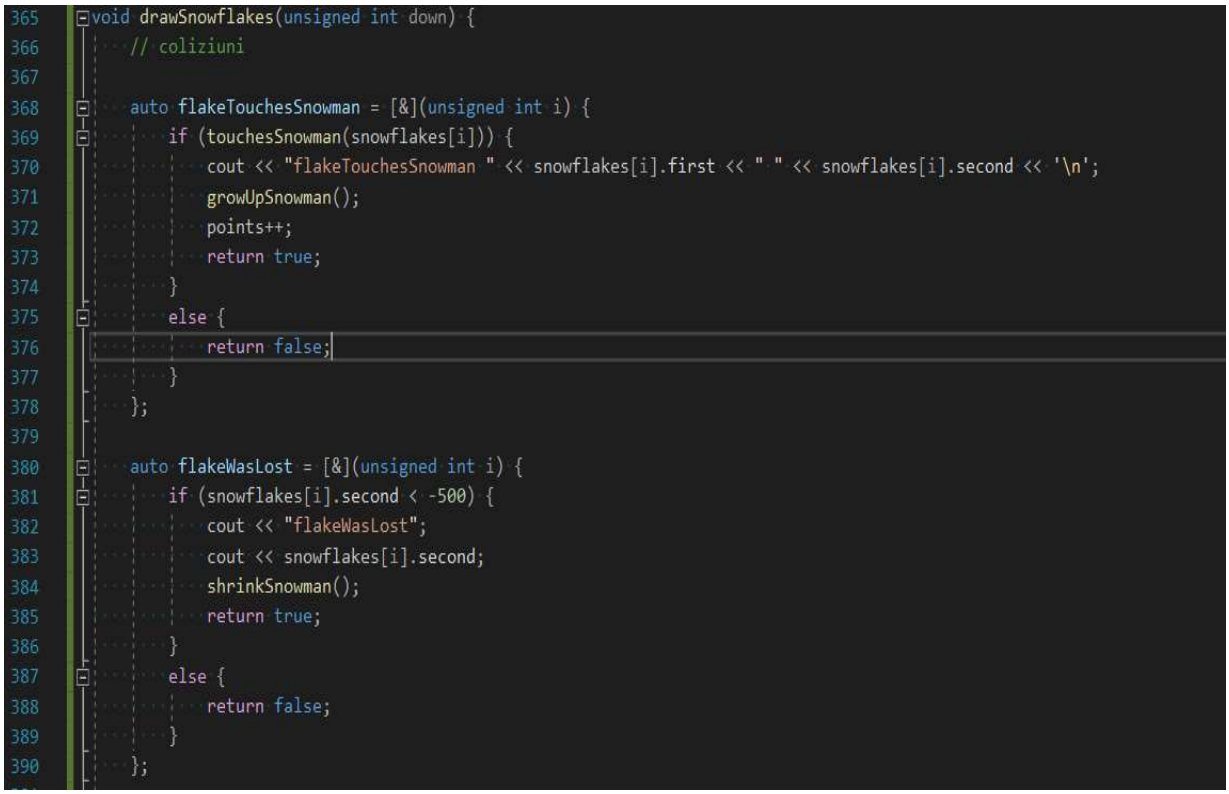
Calculez matricea de translație pentru fiecare fulg in parte în functie de parametrul transmis numit “down”.

```
392     // parcurgere fulgi
393
394     for (auto i = 0; i < snowflakes.size(); i++)
395     {
396         // calculez translatia pentru o coordonata mai mica pe axa Oy => efectul de coborare.
397         snowflakes[i].second -= down;
398         snowflakeTransl = glm::translate(glm::mat4(1.0f), glm::vec3(snowflakes[i].first, snowflakes[i].second, 0.0));
399
```

Coliziuni:

Creșterea și scăderea omului de zăpadă e controlată de coliziunile acestuia cu fulgii de zăpadă:

```
315 bool inline touchesSnowman(pair<int, int> snowflake) {
316     const float snowmanOriginalPositionOX = 200.0f;
317     const float snowmanPosition = snowmanOriginalPositionOX + tx;
318     const float headRadius = 50.0f;
319     const float bodyRadius = 75.0f;
320     const float bypass = 20.0f;
321     const float headHeight = headRadius * 2;
322     const float bodyHeight = bodyRadius * 2 - bypass;
323     const float snowmanHeight = headHeight + bodyHeight;
324     const float feetLevel = -380.0f;
325
326
327     auto touchesHead = [&]() {
328         const bool isBetweenLeftAndRightOfHead = (snowflake.first > snowmanPosition - headRadius * grow) && (snowflake.first < snowmanPosition + headRadius * grow);
329         const bool isBetweenTopAndBottomOfHead = (snowflake.second <= feetLevel + snowmanHeight * grow) && (snowflake.second > feetLevel + bodyHeight * grow);
330
331         return (isBetweenLeftAndRightOfHead && isBetweenTopAndBottomOfHead);
332     };
333
334     auto touchesBody = [&]() {
335         const bool isBetweenLeftAndRightOfBody = (snowflake.first > snowmanPosition - bodyRadius * grow) && (snowflake.first < snowmanPosition + bodyRadius * grow);
336         const bool isBetweenTopAndBottomOfBody = (snowflake.second <= feetLevel + bodyHeight * grow) && (snowflake.second > feetLevel);
337
338         return (isBetweenLeftAndRightOfBody && isBetweenTopAndBottomOfBody);
339     };
340
341     return (touchesBody() || touchesHead());
342 }
```



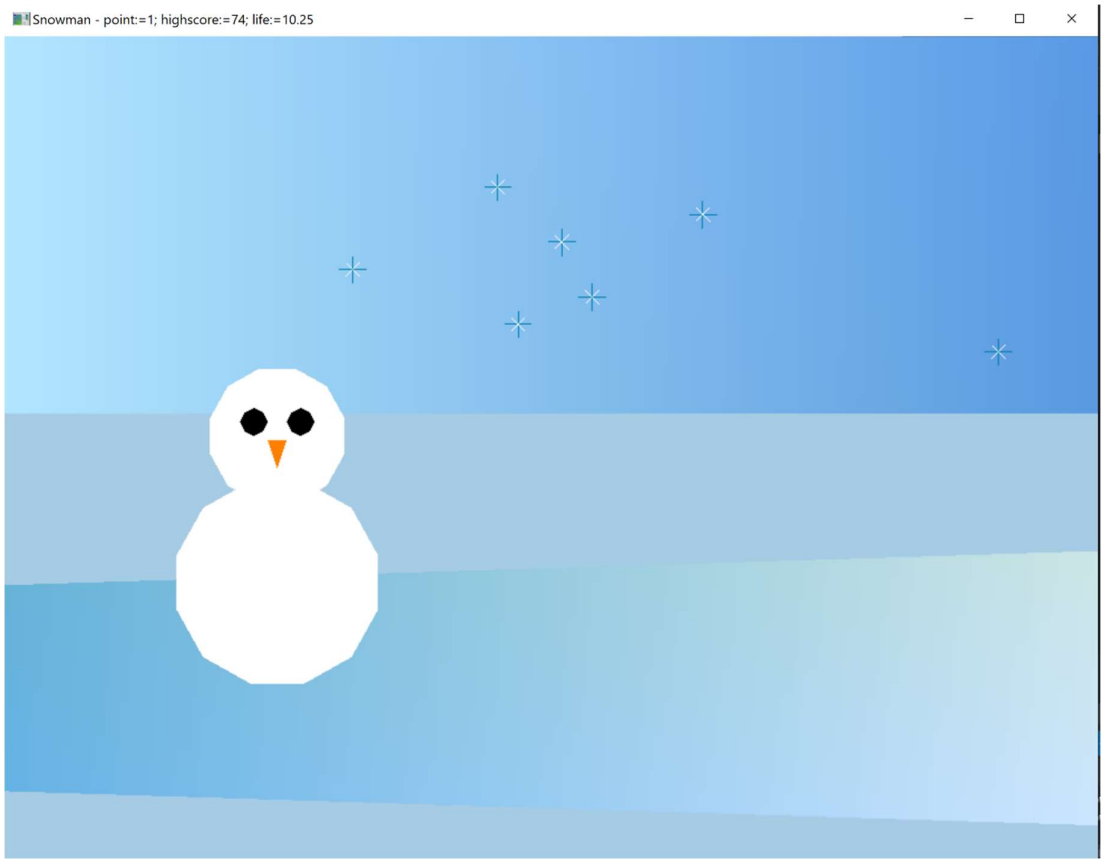
```
void drawSnowflakes(unsigned int down) {
    // coliziuni

    auto flakeTouchesSnowman = [&](unsigned int i) {
        if (touchesSnowman(snowflakes[i])) {
            cout << "flakeTouchesSnowman " << snowflakes[i].first << " " << snowflakes[i].second << '\n';
            growUpSnowman();
            points++;
            return true;
        }
        else {
            return false;
        }
    };

    auto flakeWasLost = [&](unsigned int i) {
        if (snowflakes[i].second < -500) {
            cout << "flakeWasLost";
            cout << snowflakes[i].second;
            shrinkSnowman();
            return true;
        }
        else {
            return false;
        }
    };
```

Originalitate:

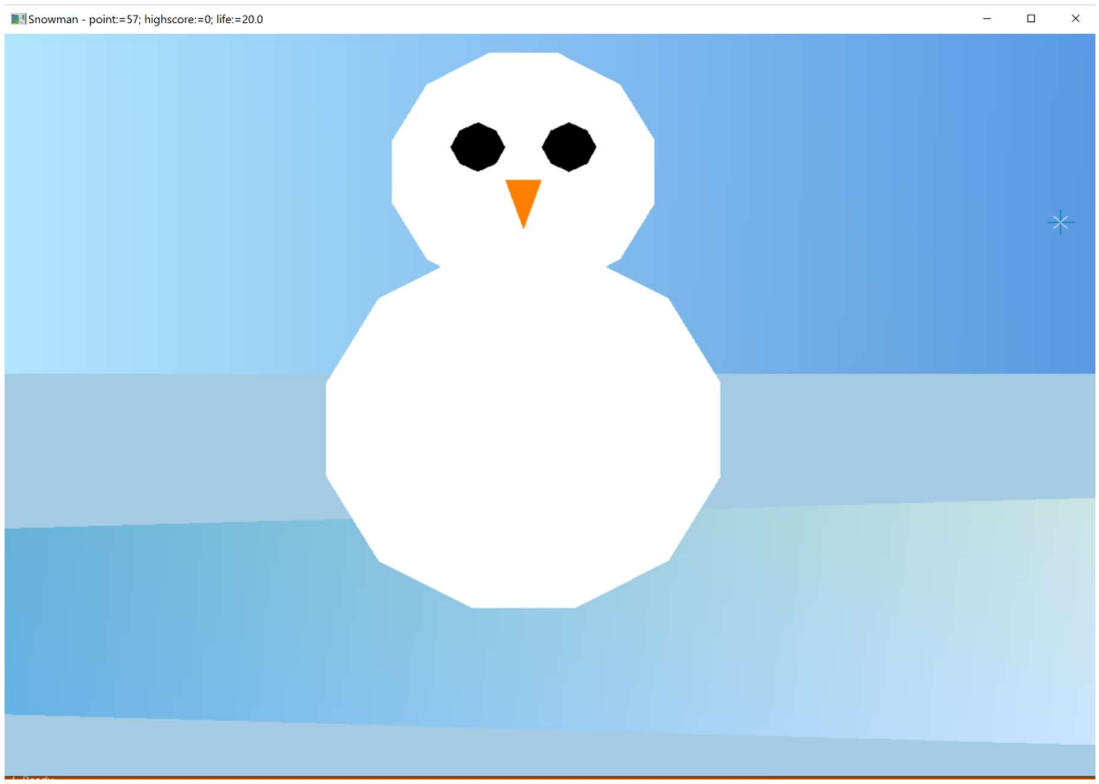
Tema și toate detaliile proiectului au fost idei proprii puse în aplicare cu ajutorul cunostiintelor acumulate la clasă. Un aspect nemenționat până acum care îl face special este înfățișarea unui peisaj de iarnă, în care patinoarul și cerul au o notă realistă datorită gradientului.

Aspectul proiectului:

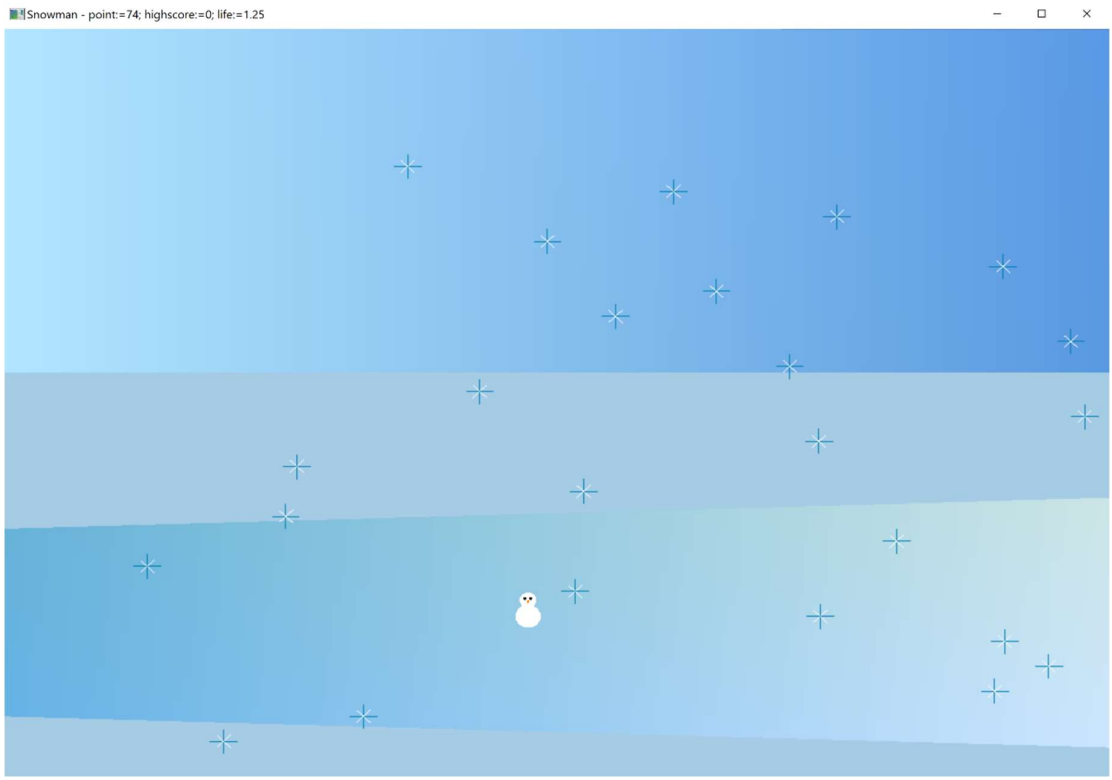
- La începerea jocului**:**



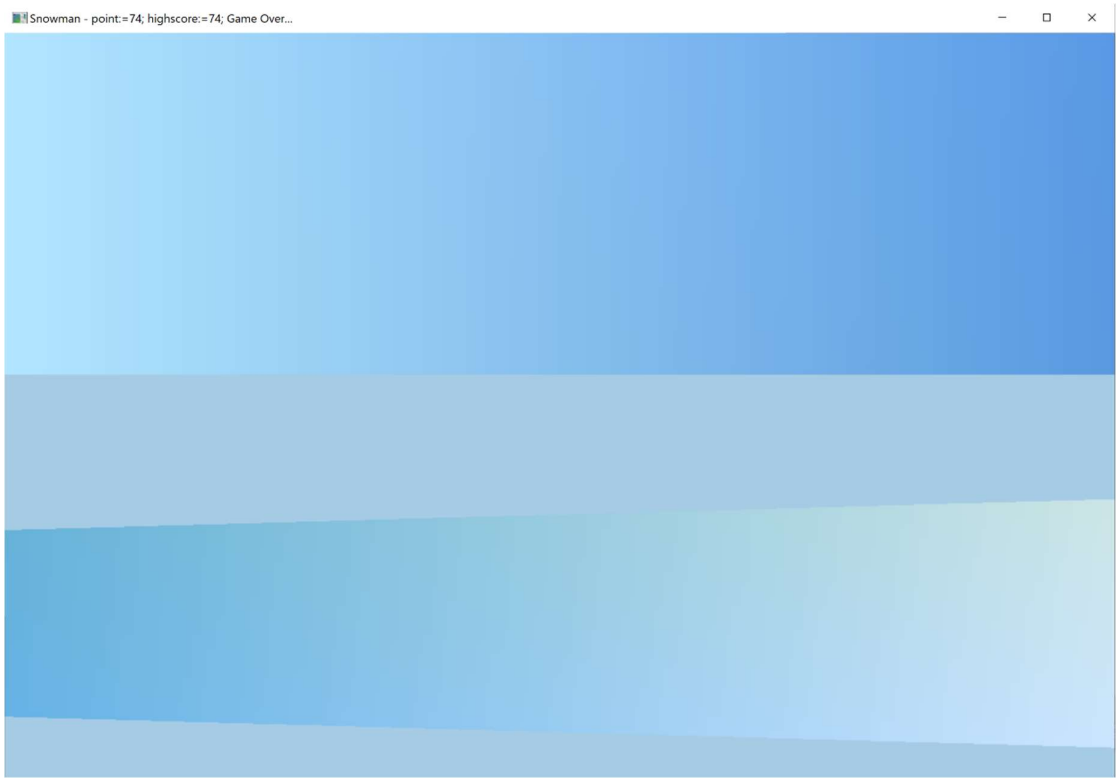
- Atunci cand omul de zăpadă a ajuns la capacitate maxima (se observa scorul de 57 de puncte acumulate, ceea ce inseamna ca a prins 57 de fulgi). Se observă că o dată atinsă această dimensiune, nu mai poate crește chiar dacă prinde și alți fulgi.



- Atunci când omul de zăpadă a pierdut prea mulți fulgi și este foarte aproape de Game Over:



- Atunci cand este Game Over:
    - omul de zăpadă s-a făcut atât de mic încât a dispărut
    - ninsoarea s-a oprit
    - a aparut mesajul “Game Over” in bara ecranului):



Contributii individuale:

Barbu Iulia**:**

- Aspectul proiectului
- Desenarea primitivelor
- Asocierea culorilor în shadere
- Mișcarea omului de zăpadă cu ajutorul tastelor
- Generarea aleatoare a vectorului de fulgi și căderea acestora
- Măsurarea timpului și sincronizarea fulgilor

Staicu Octavian**:**

- Rotirea fulgilor și oprirea acestora la momentul “Game Over”
- Verificarea coliziunii dintre omul de zăpada si fulgi
- Micșorarea si creșterea omului de zăpadă
- Etapele jocului și afișarea informațiilor în title bar: highscore, points, life, “Game Over”.
- Aranjarea codului și corectarea warningurilor
- Code refactoring + optimizarea înmulțirilor matricilor

Modificări de la prezentare:

- Creșterea/scăderea omului de zăpadă
- Gameplay: pierderea jocului când s-a topit omul de zăpadă, puncte acumulate, viață, highscore
- Rotirea fulgilor
- Schimbarea omului de zăpadă de la octogon la dodecagon
- Citibilitatea codului, refactoring, optimizat înmulțirile matricilor
- Optimizat substanțial coliziunile dintre omul de zăpadă și fulgi pentru a funcționa corespunzător, în funcție de factorul de scalare al acestuia

//snowman.cpp

```
#include <windows.h> // biblioteci care urmeaza sa fie incluse
#include <stdlib.h> // necesare pentru citirea shader-elor
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <GL/glew.h> // glew apare inainte de freeglut
#include <GL/freeglut.h> // nu trebuie uitat freeglut.h
#include "loadShaders.h"
#include "glm/glm/glm.hpp"
#include "glm/glm/gtc/matrix_transform.hpp"
#include "glm/glm/gtx/transform.hpp"
#include "glm/glm/gtc/type_ptr.hpp"
#include <string>
#include <fstream>
#include <time.h>

using namespace std;

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

GLuint
VaoId,
```

VboId,

ColorBufferId,

ProgramId,

myMatrixLocation,

matrScaleLocation,

matrTranslLocation,

matrRotlLocation,

codColLocation;

glm::mat4 myMatrix,

resizeMatrix,

moveTransl,

matrScale,

matrRot, mTest,

matrDeplasare,

eyeRightTransl,

headAllTranslates,

snowflakeTransl,

snowflakeRotate,

snowflakeTranslRot,

snowflakeTranslRotInv,

growTransl1,

growScale,

growTransl2;

```
float tx = 0, grow = 1.0f;;
const float PI = 3.141592f, growFactor = 0.025f;
time_t last_time = 0, current_time = 0, last_time2 = 0, current_time2 = 0;
vector< pair <int, int> > snowflakes; // vector de perechi unde o pereche == (x, y) din translatia aplicata fulgului initial pt a obtine unul nou.
int codCol;
unsigned int highscore = 0u, points = 0u, alpha = 0u;

void displayMatrix()
{
   for (int ii = 0; ii < 4; ii++)
   {
      for (int jj = 0; jj < 4; jj++)
         cout << myMatrix[ii][jj] << " ";
      cout << endl;
   };
   cout << "\n";
};



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

```
GLfloat Vertices[] = {
    // fundal - sky
    800.0f, 0.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 0.0f, 1.0f,
    0.0f, 600.0f, 0.0f, 1.0f,
    800.0f, 600.0f, 0.0f, 1.0f,

    // patinoar
    800.0f, 225.0f, 0.0f, 1.0f,
    0.0f, 200.0f, 0.0f, 1.0f,
    0.0f, 50.0f, 0.0f, 1.0f,
    800.0f, 25.0f, 0.0f, 1.0f,

    // snow 1
    800.0f, 25.0f, 0.0f, 1.0f,
    0.0f, 50.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 0.0f, 1.0f,
    800.0f, 0.0f, 0.0f, 1.0f,

    // snow 2
    800.0f, 325.0f, 0.0f, 1.0f,
    0.0f, 325.0f, 0.0f, 1.0f,
    0.0f, 200.0f, 0.0f, 1.0f,
    800.0f, 225.0f, 0.0f, 1.0f,
```

```
// Snowman's body
// https://www.mathopenref.com/coordpolycalc.html
219.0f,128.0f, 0.0f, 1.0f,
181.0f,128.0f, 0.0f, 1.0f,
147.0f,147.0f, 0.0f, 1.0f,
128.0f,181.0f, 0.0f, 1.0f,
128.0f,219.0f, 0.0f, 1.0f,
147.0f,253.0f, 0.0f, 1.0f,
181.0f,272.0f, 0.0f, 1.0f,
219.0f,272.0f, 0.0f, 1.0f,
253.0f,253.0f, 0.0f, 1.0f,
272.0f,219.0f, 0.0f, 1.0f,
272.0f,181.0f, 0.0f, 1.0f,
253.0f,147.0f, 0.0f, 1.0f,

// varfuri pt ochiul din stanga ( pe al doilea o sa il translatez )
165.0f, 205.0f, 0.0f, 1.0f,
160.0f, 215.0f, 0.0f, 1.0f,
165.0f, 225.0f, 0.0f, 1.0f,
175.0f, 230.0f, 0.0f, 1.0f,
185.0f, 225.0f, 0.0f, 1.0f,
190.0f, 215.0f, 0.0f, 1.0f,
185.0f, 205.0f, 0.0f, 1.0f,
```

```
    175.0f, 200.0f, 0.0f, 1.0f,

    // varfuri pt nas
    190.0f, 195.0f, 0.0f, 1.0f,
    210.0f, 195.0f, 0.0f, 1.0f,
    200.0f, 165.0f, 0.0f, 1.0f,

    // fulg cu centru de (7.5, 510)
    10.0f, 520.0f, 0.0f, 1.0f,
    10.0f, 500.0f, 0.0f, 1.0f,
    0.0f, 510.0f, 0.0f, 1.0f,
    20.0f, 510.0f, 0.0f, 1.0f,
    5.0f, 505.0f, 0.0f, 1.0f,
    15.0f, 515.0f, 0.0f, 1.0f,
    5.0f, 515.0f, 0.0f, 1.0f,
    15.0f, 505.0f, 0.0f, 1.0f,
};


GLfloat Colors[] = {
    // cerul
    0.35f, 0.6f, 0.85f, 1.0f,
    0.7f, 0.9f, 1.0f, 1.0f,
    0.7f, 0.9f, 1.0f, 1.0f,
```

```
        0.35f, 0.6f, 0.9f,1.0f,
        // patinoarul
        0.8f, 0.9f, 0.9f, 1.0f,
        0.4f, 0.7f, 0.85f, 1.0f,
        0.4f, 0.7f, 0.9f,1.0f,
        0.8f, 0.9f, 1.0f, 1.0f,
    };
    // se creeaza un buffer nou
    glGenBuffers(1, &VboId);
    // este setat ca buffer curent
    glBindBuffer(GL_ARRAY_BUFFER, VboId);
    // punctele sunt "copiate" in bufferul curent
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
    // se creeaza / se leaga un VAO (Vertex Array Object) - util cand se utilizeaza mai multe VBO
    glGenVertexArrays(1, &VaoId);
    glBindVertexArray(VaoId);
    // se activeaza lucrul cu atribute; atributul 0 = pozitie
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);
    // un nou buffer, pentru culoare
    glGenBuffers(1, &ColorBufferId);
    glBindBuffer(GL_ARRAY_BUFFER, ColorBufferId);
```

```
   glBufferData(GL_ARRAY_BUFFER, sizeof(Colors), Colors, GL_STATIC_DRAW);
   // atributul 1 = culoare
   glEnableVertexAttribArray(1);
   glVertexAttribPointer(1, 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("snowman_Shader.vert", "snowman_Shader.frag");
```

```
    glUseProgram(ProgramId);
}



void DestroyShaders(void)
{
    glDeleteProgram(ProgramId);
}



void Initialize(void)
{
    //read highscore from file, if it exists
    try {
        ifstream fin("snowman_Highscore.txt");
        if (fin)
            fin >> highscore;
        else
            cout << "snowman_Highscore.txt does not exists\n";
        fin.close();
    }
    catch (...) {
        highscore = 0;
    }
```

```
    //generator numere aleatoare
    srand(static_cast<unsigned int>(time(NULL)));


    // matrDeplasare- ma muta mai la stanga jos, o folosesc ca sa nu mai umblu in [-400, 400] x [-300, 300], ci in [0, 800] x [0, 600]
    // resizeMatrix imi schimba proportia ecranului, imi discretizeaza intervalul [-400, 400] x [-300, 300] in [-1, 1] pt a se putea realiza desenarea
    matrDeplasare = glm::translate(glm::mat4(1.0f), glm::vec3(-400.f, -300.f, 0.0));
    resizeMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(1.f / 400, 1.f / 300, 1.0));
    myMatrix = resizeMatrix * matrDeplasare;


    //rotate snowflake
    snowflakeTranslRot = glm::translate(glm::mat4(1.0f), glm::vec3(-7.5, -510, 0.0));
    snowflakeTranslRotInv = glm::translate(glm::mat4(1.0f), glm::vec3(7.5, 510, 0.0));


    // Snowman's components
    // modific pozitia si marimea "poligonului corp" cu ajutorul matricelor de scalare si translatie, ca apoi sa il folosesc drept cap
    // obtin ochiul drept translatand ochiul stang pe axa Ox
    glm::mat4 headScale = glm::scale(glm::mat4(1.0f), glm::vec3(2.f / 3, 2.f / 3, 1.0));
```

```
   glm::mat4 headTransl1 = glm::translate(glm::mat4(1.0f), glm::vec3(-200.f, -100.f, 0.0));

   glm::mat4 headTransl2 = glm::translate(glm::mat4(1.0f), glm::vec3(200.f, 100.f, 0.0));

   glm::mat4 headUpTransl = glm::translate(glm::mat4(1.0f), glm::vec3(0.f, 137.5f, 0.0));

   headAllTranslates = headUpTransl * headTransl2 * headScale * headTransl1;


   //snowman right's eye

   eyeRightTransl = glm::translate(glm::mat4(1.0f), glm::vec3(50.f, 0.f, 0.0));



   // Grow Snowman

   growTransl1 = glm::translate(glm::mat4(1.0f), glm::vec3(-200.f, -120.f, 0.0));

   growTransl2 = glm::translate(glm::mat4(1.0f), glm::vec3(200.f, 120.f, 0.0));

   growScale = glm::scale(glm::mat4(1.0f), glm::vec3(1.f, 1.f, 1.0));


   glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

   CreateVBO();

   CreateShaders();

}
```

```
void processSpecialKeys(int key, int x, int y)
{
   switch (key) {
   case GLUT_KEY_LEFT:
      // chiar daca ecranul meu e intre 0 - 800, eu am plasat omul de zapada sa stea initial undeva pe la poz 250
      // si atunci, tx == 0 atunci cand omul de zapada e la pozitia 250
      // deci d aia tx e decalat cam cu -250 fata de marginile reale ale ecranului
      if (tx <= -250)
         tx = 650;
      else
         tx -= 10.f;
      break;

   case GLUT_KEY_RIGHT:
      if (tx >= 650)
         tx = -250;
      else
         tx += 10.f;
      break;
   default:
      break;
```

```
    }
}


void generateSnowflakes()
{
    current_time = time(NULL);

    if (current_time != last_time)
    {
        last_time = current_time;
        int randX = rand() % 700 + 100;
        cout << randX << " ";
        snowflakes.push_back(make_pair(randX, 0));
    }
}


void drawSnowflake() {
    // Fulg

    //half1
    codCol = 5; //
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
```

```
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_LINES, 39, 4);

    //half2
    codCol = 2; //
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_LINES, 43, 4);
}



void writeScore() {
    if (points > highscore) {
        highscore = points;
        ofstream fout("snowman_Highscore.txt");
        fout << points;
        fout.close();
    }
}



bool inline touchesSnowman(pair<int, int> snowflake) {
    const float snowmanOriginalPositionOX = 200.0f;
    const float snowmanPosition = snowmanOriginalPositionOX + tx;
```

```
const float headRadius = 50.0f;
const float bodyRadius = 75.0f;
const float bypass = 20.0f;
const float headHeight = headRadius * 2;
const float bodyHeight = bodyRadius * 2 - bypass;
const float snowmanHeight = headHeight + bodyHeight;
const float feetLevel = -380.0f;


auto touchesHead = [&]() {
    const bool isBetweenLeftAndRightOfHead = (snowflake.first > snowmanPosition - headRadius * grow) && (snowflake.first < snowmanPosition + headRadius * grow);
    const bool isBetweenTopAndBottomOfHead = (snowflake.second <= feetLevel + snowmanHeight * grow) && (snowflake.second > feetLevel + bodyHeight * grow);

    return (isBetweenLeftAndRightOfHead && isBetweenTopAndBottomOfHead);
};

auto touchesBody = [&]() {
    const bool isBetweenLeftAndRightOfBody = (snowflake.first > snowmanPosition - bodyRadius * grow) && (snowflake.first < snowmanPosition + bodyRadius * grow);
```

```
        const bool isBetweenTopAndBottomOfBody = (snowflake.second <= feetLevel + bodyHeight * grow) && (snowflake.second > feetLevel);

        return (isBetweenLeftAndRightOfBody && isBetweenTopAndBottomOfBody);
    };

    return (touchesBody() || touchesHead());
}

void growUpSnowman() {
    grow += growFactor;
    cout << "\ngrow_up_snowman: " << grow << "\n";
    if (grow >= 2.0f)
        grow = 2.0f;
    growScale = glm::scale(glm::mat4(1.0f), glm::vec3(grow, grow, 1.0));
}

void shrinkSnowman() {
    grow -= growFactor * 4;
    if (grow < 0.0f) {
        grow = 0.0f;
```

```
    }
    cout << "\nshrink_snowman: " << grow << "\n";

    growScale = glm::scale(glm::mat4(1.0f), glm::vec3(grow, grow, 1.0));
}



void drawSnowflakes(unsigned int down) {
    // coliziuni

    auto flakeTouchesSnowman = [&](unsigned int i) {
        if (touchesSnowman(snowflakes[i])) {
            cout << "flakeTouchesSnowman " << snowflakes[i].first << " " << snowflakes[i].second << '\n';
            growUpSnowman();
            points++;
            return true;
        }
        else {
            return false;
        }
    };

    auto flakeWasLost = [&](unsigned int i) {
```

```
    if (snowflakes[i].second < -500) {
        cout << "flakeWasLost";
        cout << snowflakes[i].second;
        shrinkSnowman();
        return true;
    }
    else {
        return false;
    }
};

// parcurgere fulgi

for (auto i = 0; i < snowflakes.size(); i++)
{
    // calculez translatia pentru o coordonata mai mica pe axa Oy => efectul de coborare.
    snowflakes[i].second -= down;
    snowflakeTransl = glm::translate(glm::mat4(1.0f), glm::vec3(snowflakes[i].first, snowflakes[i].second, 0.0));

    if (flakeTouchesSnowman(i) || flakeWasLost(i)) {
        // eliminam din memorie fulgii care fie au fost absorbiti de omul de zapada, fie au iesit din ecran
        snowflakes[i] = snowflakes.back();
```

```
            snowflakes.pop_back();
            i--;
            continue;    //fulgul a disparut, deci nu il mai desenam
        }

        // fulg cu centru de (7.5, 510)
        glm::mat4 rotate_aux = snowflakeTranslRotInv * snowflakeRotate * snowflakeTranslRot; //put the base snowflake in center of screen, rotate it, and put it back
        glm::mat4 rotateSnowflake = resizeMatrix * matrDeplasare * snowflakeTransl * rotate_aux;// rotate the snowflake, put it in the right place on the screen, then size the screen
        myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
        glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &rotateSnowflake[0][0]);
        drawSnowflake();
    }
}


void RenderFunction(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    myMatrix = resizeMatrix * matrDeplasare;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
```

```
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

// cerul
codCol = 0; // 0 == gradient
codColLocation = glGetUniformLocation(ProgramId, "codCol");
glUniform1i(codColLocation, codCol);
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

// patinoarul
codCol = 0;
codColLocation = glGetUniformLocation(ProgramId, "codCol");
glUniform1i(codColLocation, codCol);
glDrawArrays(GL_TRIANGLE_FAN, 4, 4);

// zapada de jos
codCol = 1; // 1 == snow color
codColLocation = glGetUniformLocation(ProgramId, "codCol");
glUniform1i(codColLocation, codCol);
glDrawArrays(GL_TRIANGLE_FAN, 8, 4);

// zapada din spate
glUniform1i(codColLocation, codCol);
glDrawArrays(GL_TRIANGLE_FAN, 12, 4);
```

```
    if (grow > 0.0f)
        generateSnowflakes();

    current_time2 = time(NULL);

    if (current_time2 != last_time2)
    {
        last_time2 = current_time2;
        snowflakeRotate = glm::rotate_slow(glm::mat4(1.0f), (++alpha) * PI / 8, glm::vec3(0.0, 0.0, 1.0));
        drawSnowflakes(20);
    }
    else
        drawSnowflakes(0);

    //// grow and move Snowman ( if needed)
    moveTransl = glm::translate(glm::mat4(1.0f), glm::vec3(tx, 0.f, 0.0));
    glm::mat4 growAux = growTransl2 * growScale * growTransl1;
    myMatrix = myMatrix * moveTransl * growAux;
    // se duce in programul nostru pe care l am definit ca fiind compus din cele 2 shadere 04_03 stocate in GPU
    // imi gaseste variabila uniforma myMatrix (declarata in 04_03_Shader.vert)
    // si asa imi returneaza locatia din GPU
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
```

```
    // acum la locatia respectiva suprascriu cu o singura matrice (parametrul
1), si anume
    // matricea stocata de aici din CPU, &myMatrix[0][0]
    // GL_FALSE inseamna ca nu vreau sa o transpuna sau cv de genu
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

    //// Snowman's body
    codCol = 2; // 2 == white
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_TRIANGLE_FAN, 16, 12);

    // Snowman's head
    myMatrix = myMatrix * headAllTranslates;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_TRIANGLE_FAN, 16, 12);

    //// Snowman's left eye
    codCol = 3; // 3 == black
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_TRIANGLE_FAN, 28, 8);
```

```
   // Snowman's nose
   codCol = 4; // 4 == orange
   codColLocation = glGetUniformLocation(ProgramId, "codCol");
   glUniform1i(codColLocation, codCol);
   glDrawArrays(GL_TRIANGLE_FAN, 36, 3);

   // Snowman's right eye
   codCol = 3; //
   codColLocation = glGetUniformLocation(ProgramId, "codCol");
   glUniform1i(codColLocation, codCol);
   myMatrix = myMatrix * eyeRightTransl;
   myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
   glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
   glDrawArrays(GL_TRIANGLE_FAN, 28, 8);

   // scor
   string titlu = "Snowman - point:=" + to_string(points) + "; highscore:=" +
to_string(highscore);

   if (grow > 0.0f) {
      int grow_int = lround(grow * 1000);
      titlu += "; life:=" + to_string(grow_int / 100) + "." + to_string(grow_int
% 100);
   }
```

```
    else {
        titlu += "; Game Over...";
        writeScore();
    }

    glutSetWindowTitle(titlu.c_str());

    glutSwapBuffers();
    glFlush();
}


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


int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition(0, 0);
```

```
    glutInitWindowSize(800, 600);
    glutCreateWindow("Snowman");
    glewInit();
    Initialize();
    glutDisplayFunc(RenderFunction);
    glutSpecialFunc(processSpecialKeys);
    glutIdleFunc(RenderFunction);
    glutCloseFunc(Cleanup);
    glutMainLoop();
    return 0;
}
```

```
// snowman_Shader.frag
// Shader-ul de fragment / Fragment shader

 #version 400

in vec4 ex_Color;
uniform int codCol;

out vec4 out_Color;

void main(void)
  {
       if ( codCol==0 )
              out_Color = ex_Color;
       if ( codCol==1 )
              out_Color=vec4 (0.65, 0.8, 0.9, 0.0);
       if ( codCol==2 )
              out_Color=vec4 (1.0, 1.0, 1.0, 0.0);
              //out_Color=vec4 (0.4, 0.6, 0.85, 0.0);
       if ( codCol==3 )
              out_Color=vec4 (0.0, 0.0, 0.0, 0.0);
       if ( codCol==4 )
              out_Color=vec4 (1.0, 0.5, 0.0, 0.0);

  }
```

```
// snowman_Shader.vert
// Shader-ul de varfuri

 #version 400


in vec4 in_Position;
in vec4 in_Color;

out vec4 gl_Position;
out vec4 ex_Color;
uniform mat4 myMatrix;



void main(void)
  {
    gl_Position = myMatrix*in_Position;
    //gl_Position = resizeMatrix * matrDeplasare * matrTransl2iv * myScale * matrTransl1iv *
in_Position;
    ex_Color = in_Color;
   }
```

Resurse utilizate:

1. https://www.mathopenref.com/coordpolycalc.html
2. Fișierele și codurile sursă puse la dispoziție în cadrul cursului și laboratorului