

PONG în OpenGL

Acatrinei Carmen-Lorena





Brînză Tudor-Alexandru

Facultatea de Matematică și Informatică

Universitatea din București

2022

Cuprins

 Alegerea temei.....	3
 Setări inițiale.....	4
 Implementare 	5
Desenarea background-ului, a bordurii și a playerilor	5
Desenarea pătratului de joc și rotația inițială în jurul propriului centru	5
Deplasarea playerilor	6
Lovirea pătratului	6
Deplasarea pătratului de joc și obținerea punctului	6
Metode	6
1. void invarte(void).....	6
2. void mouse(int button, int state, int x, int y)	7
3. void misca1(unsigned char key, int x, int y)	7
4. void misca2(int key, int x, int y)	7
5. int loverePlayer1(void).....	7
6. int loverePlayer2(void).....	7
7. void miscaPatrat(void).....	7
8. void sleep(float sec).....	8
Contribuții individuale	9
Anexe	10
main.cpp	10
Pong_Shader.frag	18
Pong_Shader.vert	18

👤 Alegerea temei

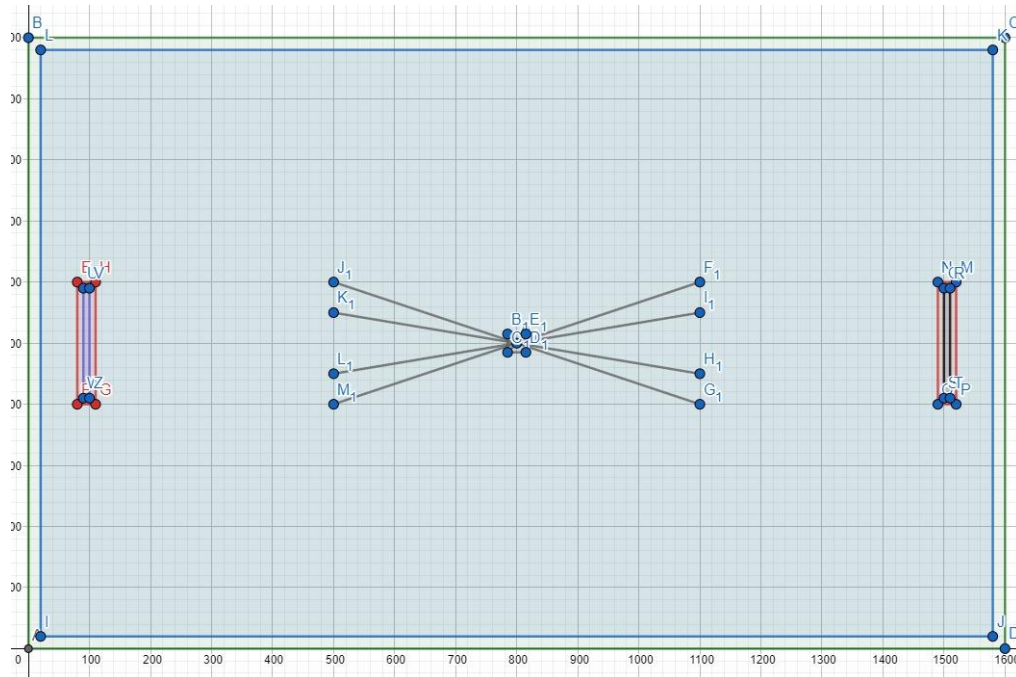
În urma rezolvării laboratoarelor și descoperirea funcționalităților din OpenGL, un mini-game părea o provocare. Ideea de Pong a venit în urma vizionării filmului *Wall-E*, conform gif-ului de mai jos:



Am început prin a juca efectiv o varianta existentă pe internet disponibilă [aici](#) pentru a înțelege mai clar ce avem de implementat și cum funcționează jocul.

♣ Setări inițiale

Primul lucru făcut a fost să ne creăm tabla de joc în [GeoGebra](#) pentru a stabili dimensiunile spațiului și obiectelor de joc.



Ulterior, am luat pas cu pas componentele, de la background până la cei doi playeri și pătratul de joc și am hotărât culorile care păreau să se potrivească.

👤 Implementare 👤

Desenarea background-ului, a bordurii și a playerilor

Am hotărât să păstrăm background-ul negru și bordura verde și să luăm dimensiunea tablei de joc de 1600x1000 cu bordura de 20.0.

Poziția playerilor este cu centrul la 60.0 stânga (pentru player 1)/dreapta (pentru player 2) față de bordură, iar dimensiunea este de 30.0x160.0. Pentru dreptunghiul colorat din interior am luat distanța egală stânga-dreapta sus-jos de 10.0, rezultând o dimensiune de 10.0x140.0.

Desenarea pătratului de joc și rotația inițială în jurul propriului centru

Centrul pătratului alb de joc se află în centrul tablei de joc, mai exact la 800x500. Dimensiunea pătratului este de 30.0x30.0. După ce am poziționat pătratul, am aplicat rotație în jurul propriului centru aplicând translație în origine, rotație și translație înapoi la poziția inițială și ne-am folosit de metoda idle [invarste](#), descrisă în secțiunea [Metode](#).

```
393     matrRot = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0, 0.0, 1.0));
394     // Translatia 1: centrul patratului este translatat
395     matrTransl1 = glm::translate(glm::mat4(1.0f), glm::vec3(-800.f, -500.f, 0.0));
396     // Translatia 2: inversa/opusa translatiei T1
397     matrTransl2 = glm::translate(glm::mat4(1.0f), glm::vec3(800.f, 500.f, 0.0));
```

Rotația pătratului începe odată cu randarea ecranului și se oprește la primul click stânga, moment în care începe jocul în sine.

```
433     if (click == 0){
434         myMatrix = resizeMatrix * matrTransl2 * matrRot * matrTransl1;
435         glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
436
437         codCol = 2;
438         glUniform1i(codColLocation, codCol);
439         glDrawArrays(GL_POLYGON, 20, 4);
440     }
```

Pentru aceasta, am luat o variabilă `click` ce avea inițial valoarea 0 și ne-am folosit de funcția [mouse](#) pentru a-i modifica valoarea în 1 și pentru a stabili direcția de plecare.

Modificare în urma discuției din cadrul laboratorului:

Am adăugat și o scalare la început, în timpul rotației pătratului în jurul propriului centru. Astfel, până la apăsarea click stânga, pătratul este de 2.5 ori mai mare decât dimensiunea sa din

timpul jocului.

```
433 myMatrix = resizeMatrix * matrTransl2 * matrScale * matrRot * matrTransl1;|
434 glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
435
```

Deplasarea playerilor

Pentru a deplasa Player1 (cel din stânga) ne folosim de tastele W-S, iar pentru Player2 (cel din dreapta) ne folosim de tastele speciale UP-DOWN, folosind metodele [misca1](#) și [misca2](#).

Lovirea pătratului

În momentul în care pătratul de joc se află în zona unuia dintre playeri, acesta își schimbă direcția de deplasare și jocul continuă. Lovirea acestuia de Player1 se realizează cu ajutorul metodei [lovirePlayer1](#), iar pentru cel de-al doilea Player avem metoda [lovirePlayer2](#). Modul în care se realizează lovirea este explicat pe larg în secțiunea [Metode](#).

Deplasarea pătratului de joc și obținerea punctului

Atât pentru deplasarea pătratului de joc, cât și pentru obținerea punctului (un punct se obține în momentul în care pătratul de joc nu este lovit de către Player și depășește marginea laterală) ne-am folosit de metoda [miscaPatrat](#). Pătratul își începe deplasarea în momentul apăsării click stânga (când variabila `click == 1` și `punctMarcat == 0`), iar în momentul marcării unui punct (`punctMarcat == 1`) afișăm scorul, îi atribuim variabilei `punctMarcat` valoarea 0 și apelăm metoda [sleep](#) pentru a bloca tabla de joc timp de 2 secunde.

Metode

1. `void invarte(void)`

Metoda are rolul de a modifica unghiul `angle` pentru a realiza rotația pătratului din centru prin scăderi cu `beta` (inițializat cu `0.009`, valoare aleasă în urma mai multor încercări pentru a avea o viteză potrivită). Această modificare se realizează doar dacă nu a fost încă

apăsător `GLUT_LEFT_BUTTON` (`click == 0`). Dacă a fost apăsător `GLUT_LEFT_BUTTON`, `angle` nu mai este modificat, aşadar rotaţia se opreşte.

2. `void mouse(int button, int state, int x, int y)`

Metoda primeşte parametrii standard, dintre care noi ne folosim doar de `button` pentru a recunoaşte butonul apăsător (în cazul nostru, `GLUT_LEFT_BUTTON`). În momentul apăsătorii acestui buton, valoarea variabilei `click` se modifică în 1 şi se stabileşte direcţia de plecare.

3. `void misca1(unsigned char key, int x, int y)`

În cadrul acestei metode ne-am folosit de

```
#define KEY_DOWN( vk_code ) ( ( GetAsyncKeyState( vk_code ) & 0x8000 ) ? 1 : 0 ) ,
```

care primeşte ca parametru codul unei taste şi identifică dacă această tastă este apăsătoră sau nu.

În momentul apăsătorii uneia dintre taste (UP, DOWN, W sau S), am verificat dacă variabilele `ty1` (pentru Player1), respectiv `ty2` (pentru Player2) depăşesc limitele tablei de joc (395.0 şi -395.0) şi am modificat valoarea acestora adăugând sau scăzând 20.0, după caz.

4. `void misca2(int key, int x, int y)`

Această metodă este aproape identică cu cea de mai sus (`misca1`) şi diferă prin tipul parametrului `key` şi prin apel. (`misca1` este argument al funcţiei `glutKeyboardFunc()`, în timp ce `misca2` este argument al funcţiei `glutSpecialFunc()`).

5. `int lovirePlayer1(void)`

În cadrul acestei metode am verificat dacă coordonatele pătratului de joc (`i` şi `j`) sunt cuprinse total de către Player1 (`j + 15 <= ty1 + 80 && j - 15 >= ty1 - 80`) sau dacă pătratul se află cu jumătatea inferioară (respectiv superioară) deasupra (respectiv sub) Player1 şi cea superioară (respectiv inferioară) sub (respectiv deasupra) acestuia. Metoda returnează 1 pentru situaţiile descrise anterior şi 0, altfel.

6. `int lovirePlayer2(void)`

Această metodă funcţionează la fel cu `lovirePlayer1`, doar că utilizează variabila `ty2` pentru a aplica verificările celui de-al doilea Player.

7. `void miscaPatrat(void)`

În cadrul acestei metode ne-am folosit de variabilele:

- `i` şi `j`, reprezentând coordonatele pătratului;

- `directie`, având valorile 0 pentru dreapta sus, 1 pentru dreapta jos, 2 pentru stânga jos și 3 pentru stânga sus și reprezentând direcția din care vine pătratul de joc în momentul lovirii de una dintre margini (decisă de către `i` și `j`);
- `ty1` și `ty2`, reprezentând înălțimea celor doi playeri pe tabla de joc;
- `punctMarcat`, având valoarea 0 cât timp pătratul nu a depășit marginile laterale ale tablei de joc și 1 altfel;
- `scorP1` și `scorP2`, memorând scorul celor doi playeri.

În funcție de `directie`, modificăm coordonatele pătratului (`i` și `j`) cu 0.5, valoare aleasă pentru a avea o viteză potrivită de deplasare.

Prima direcție de plecare este aleasă random (una dintre cele 4 valori ale variabilei `directie`).

Schimbarea direcției are loc atunci când pătratul lovește marginea superioară sau cea inferioară sau când lovește unul dintre playeri. Modificarea coordonatelor după lovire este condiționată de direcția pătratului dinainte de lovire.

Un punct este obținut în momentul în care coordonata `i` a pătratului depășește valoarea -830 sau 830, după caz. În momentul obținerii unui punct, pozițiile playerilor și coordonatele pătratului se resetează la valorile inițiale (`i = 0`; `j = 0`; `ty1 = 0`, `ty2 = 0`), `punctMarcat` devine 1, iar direcția de plecare a pătratului este aleasă random către playerul care a marcat punctul. Valoarea `scorP1` (sau `scorP2`) este incrementată, după caz.

8. `void sleep(float sec)`

Aceasta este o metodă ajutătoare pentru a întrerupe activitatea ecranului timp de `sec` secunde.

🧠 Contribuții individuale

Ideile pentru proiect au fost gândite și puse împreună de către ambii membrii ai echipei. Contribuțiile au fost aduse în mod egal și comunicarea a fost una naturală, ne-am completat fiecare propunere și am lucrat într-un mod eficient. Modul de lucru a fost unul armonios și nu au existat contradicții, iar fiecare modificare a fost făcută de comun acord.

Anexe

main.cpp

```
/* PONG.
*/
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <time.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"

#define KEY_DOWN( vk_code ) ( ( GetAsyncKeyState( vk_code ) & 0x8000 ) ? 1 : 0 )
// tasta apasata

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

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

glm::mat4
myMatrix, resizeMatrix, matrTransl, matrTransl1, matrTransl2, matrTranslP1,
matrTranslP2, matrTranslMiscare, matrTranslMiscare2, matrScale, matrRot, mTest;

int codCol;
float width = 1600, height = 1000;
float PI = 3.141592, angle = 0;
float ty1 = 0; float ty2 = 0;
float i = 0.0, j = 0.0, alpha = 0.0, step = 0.0, beta = 0.009;
int click = 0, directie = 0, punctMarcat = 0;
int scorP1 = 0, scorP2 = 0;

void sleep(float sec)
{
    float end = clock() / CLOCKS_PER_SEC + sec;
    while ((clock() / CLOCKS_PER_SEC) < end);
}

void invarte(void)
{
    if (click == 0)
        angle = angle - beta;
    if (click == 1)
```

```

        glutPostRedisplay();
    }

void misca1(unsigned char key, int x, int y)
{
    if (KEY_DOWN(38)) {
        if (ty2 < 395.0) {
            ty2 += 20;
        }
    }
    if (KEY_DOWN(40)) {
        if (ty2 > -395.0) {
            ty2 -= 20;
        }
    }
    if (KEY_DOWN(87)) {
        if (ty1 < 395.0) {
            ty1 += 20;
        }
    }
    if (KEY_DOWN(83)) {
        if (ty1 > -395.0) {
            ty1 -= 20;
        }
    }
}

void misca2(int key, int x, int y)
{
    if (KEY_DOWN(38)) {
        if (ty2 < 395.0) {
            ty2 += 20;
        }
    }
    if (KEY_DOWN(40)) {
        if (ty2 > -395.0) {
            ty2 -= 20;
        }
    }
    if (KEY_DOWN(87)) {
        if (ty1 < 395.0) {
            ty1 += 20;
        }
    }
    if (KEY_DOWN(83)) {
        if (ty1 > -395.0) {
            ty1 -= 20;
        }
    }
}

int loviPlayer1(void)
{
    if (j + 15 <= ty1 + 80 && j - 15 >= ty1 - 80) //patratul este cuprins total
    de P1
    {
        return 1;
    }
    if (j - 15 <= ty1 + 80 && j + 15 >= ty1 + 80) //patratul se afla cu
    jumatarea inferioara in dreptul lui P1 si cea superioara deasupra lui
    {
        return 1;
    }
}

```

```

    }
    if (j + 15 >= ty1 - 80 && j - 15 <= ty1 - 80) //patratul se afla cu
jumatatea superioara in dreptul lui P1 si cea inferioara sub
    {
        return 1;
    }
    return 0;
}

int lovirePlayer2(void)
{
    if (j + 15 <= ty2 + 80 && j - 15 >= ty2 - 80) //patratul este cuprins total
de P2
    {
        return 1;
    }
    if (j - 15 <= ty2 + 80 && j + 15 >= ty2 + 80) //patratul se afla cu
jumatatea inferioara in dreptul lui P2 si cea superioara deasupra lui
    {
        return 1;
    }
    if (j + 15 >= ty2 - 80 && j - 15 <= ty2 - 80) //patratul se afla cu
jumatatea superioara in dreptul lui P2 si cea inferioara sub
    {
        return 1;
    }
    return 0;
}void miscaPatrat(void)
{
    if (directie == 0)
    {
        i += 0.5;
        j += 0.5;
    }
    else if (directie == 1)
    {
        i += 0.5;
        j -= 0.5;
    }
    else if (directie == 2)
    {
        i -= 0.5;
        j -= 0.5;
    }
    else if (directie == 3)
    {
        i -= 0.5;
        j += 0.5;
    }

    //schimbarea directiei
    if (j >= 465 && directie == 0) //lovire sus venind din stanga
    {
        directie = 1; //0 sus dreapta, 1 jos dreapta, 2 jos stanga, 3 sus
stanga
    }
    else if (j >= 465 && directie == 3) //lovire sus venind din dreapta
    {
        directie = 2;
    }
    else if (j <= -465 && directie == 1) //lovire jos venind din stanga
    {
        directie = 0;
    }
}

```

```

    }
    else if (j <= -465 && directie == 2) //lovire jos venind din dreapta
    {
        directie = 3;
    }
    else if (i <= -675 && i >= -700 && directie == 2 && lovirePlayer1())
//lovire P1 venind de sus
    {
        directie = 1;
    }
    else if (i <= -675 && i >= -700 && directie == 3 && lovirePlayer1())
//lovire P1 venind de jos
    {
        directie = 0;
    }
    else if (i >= 675 && i <= 700 && directie == 1 && lovirePlayer2()) //lovire
P1 venind de sus
    {
        directie = 2;
    }
    else if (i >= 675 && i <= 700 && directie == 0 && lovirePlayer2()) //lovire
P1 venind de jos
    {
        directie = 3;
    }
}

//obtinere punct
else if (i >= 830 && directie == 1) //lovire dreapta venind de sus
{
    i = 0;
    j = 0;
    ty1 = 0;
    ty2 = 0;
    punctMarcat = 1;
    srand(time(NULL));
    int val = rand() % 2;
    if (val == 0)
    {
        directie = 2;
    }
    else
    {
        directie = 3;
    }
    scorP1++;
}
else if (i >= 830 && directie == 0) //lovire dreapta venind de jos
{
    i = 0;
    j = 0;
    ty1 = 0;
    ty2 = 0;
    punctMarcat = 1;
    srand(time(NULL));
    int val = rand() % 2;
    if (val == 0)
    {
        directie = 2;
    }
    else
    {
        directie = 3;
    }
}

```

```

        scorP1++;
    }
    else if (i <= -830 && directie == 3) //lovire stanga venind de jos
    {
        i = 0;
        j = 0;
        ty1 = 0;
        ty2 = 0;
        punctMarcat = 1;
        srand(time(NULL));
        int val = rand() % 2;
        if (val == 0)
        {
            directie = 0;
        }
        else
        {
            directie = 1;
        }
        scorP2++;
    }
    else if (i <= -830 && directie == 2) //lovire stanga venind de sus
    {
        i = 0;
        j = 0;
        ty1 = 0;
        ty2 = 0;
        punctMarcat = 1;
        srand(time(NULL));
        int val = rand() % 2;
        if (val == 0)
        {
            directie = 0;
        }
        else
        {
            directie = 1;
        }
        scorP2++;
    }
}

void mouse(int button, int state, int x, int y)
{
    switch (button) {
        case GLUT_LEFT_BUTTON && click == 0:
            click = 1;
            srand(time(NULL));
            directie = rand() % 4;
            glutIdleFunc(miscaPatrat);
            break;
        default:
            break;
    }
}

void CreateVB0(void)
{
    // varfurile
    GLfloat Vertices[] = {
        //varfuri pentru tabla de joc
        20.0f, 980.0f, 0.0f, 1.0f,
        1580.0f, 980.0f, 0.0f, 1.0f,

```

```

1580.0f, 20.0f, 0.0f, 1.0f,
20.0f, 20.0f, 0.0f, 1.0f,

//varfuri pentru player 1 (stanga)
80.0f, 420.0f, 0.0f, 1.0f,
110.0f, 420.0f, 0.0f, 1.0f,
110.0f, 580.0f, 0.0f, 1.0f,
80.0f, 580.0f, 0.0f, 1.0f,

//varfuri pentru player 2 (dreapta)
1490.0f, 580.0f, 0.0f, 1.0f,
1520.0f, 580.0f, 0.0f, 1.0f,
1520.0f, 420.0f, 0.0f, 1.0f,
1490.0f, 420.0f, 0.0f, 1.0f,

//varfuri pentru dreptunghi interior player 1 (stanga)
90.0f, 570.0f, 0.0f, 1.0f,
100.0f, 570.0f, 0.0f, 1.0f,
100.0f, 430.0f, 0.0f, 1.0f,
90.0f, 430.0f, 0.0f, 1.0f,

//varfuri pentru dreptunghi interior player 2 (dreapta)
1500.0f, 570.0f, 0.0f, 1.0f,
1510.0f, 570.0f, 0.0f, 1.0f,
1510.0f, 430.0f, 0.0f, 1.0f,
1500.0f, 430.0f, 0.0f, 1.0f,

//patratul de joc
785.0f, 515.0f, 0.0f, 1.0f,
815.0f, 515.0f, 0.0f, 1.0f,
815.0f, 485.0f, 0.0f, 1.0f,
785.0f, 485.0f, 0.0f, 1.0f,

};

GLfloat Colors[] = {
    1.0f, 0.0f, 0.0f, 1.0f,
    0.0f, 1.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f,
    1.0f, 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);
glGenBuffers(1, &ColorBufferId);
glBindBuffer(GL_ARRAY_BUFFER, ColorBufferId);
glBufferData(GL_ARRAY_BUFFER, sizeof(Colors), Colors, GL_STATIC_DRAW);
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("Pong_Shader.vert", "Pong_Shader.frag");
    glUseProgram(ProgramId);
}

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

void Initialize(void)
{
    glClearColor(0.5f, 1.0f, 0.5f, 1.0f); // culoarea de fond a ecranului
    CreateVBO();
    CreateShaders();
    codColLocation = glGetUniformLocation(ProgramId, "codCuloare");
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
}

void RenderFunction(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    resizeMatrix = glm::ortho(0.0f, width, 0.0f, height);
    matrRot = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0, 0.0, 1.0));
    // Translatia 1: centrul patratului este traslatat
    matrTransl1 = glm::translate(glm::mat4(1.0f), glm::vec3(-800.f, -500.f,
0.0));
    // Translatia 2: inversa/opusa translatiei T1
    matrTransl2 = glm::translate(glm::mat4(1.0f), glm::vec3(800.f, 500.f,
0.0));
    matrTranslP1 = glm::translate(glm::mat4(1.0f), glm::vec3(0, ty1, 0.0));
    matrTranslP2 = glm::translate(glm::mat4(1.0f), glm::vec3(0, ty2, 0.0));
    matrTranslMiscare = glm::translate(glm::mat4(1.0f), glm::vec3(i, j, 0.0));
    matrScale = glm::scale(glm::mat4(1.0f), glm::vec3(2.5, 2.5, 0.0));

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

    //tabla de joc
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_POLYGON, 0, 4);

    //player 1
    myMatrix = resizeMatrix * matrTranslP1 ;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    codCol = 2;
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_POLYGON, 4, 4);

    codCol = 3;
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_POLYGON, 12, 4);

    //player 2
    myMatrix = resizeMatrix * matrTranslP2 ;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
}

```



```

codCol = 2;
glUniform1i(codColLocation, codCol);
glDrawArrays(GL_POLYGON, 8, 4);

codCol = 4;
glUniform1i(codColLocation, codCol);
glDrawArrays(GL_POLYGON, 16, 4);

//patratul de joc
if (click == 0){
    myMatrix = resizeMatrix * matrTransl2 * matrScale * matrRot *
matrTransl1;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

    codCol = 2;
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_POLYGON, 20, 4);
}
else if (click == 1 && punctMarcat == 0) {
    myMatrix = resizeMatrix * matrTranslMiscare;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    codCol = 2;
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_POLYGON, 20, 4);
}
else if (punctMarcat == 1)
{
    punctMarcat = 0;
    std::cout << " " << scorP1 << " " << scorP2 << "\n";
    sleep(2);
}

glutPostRedisplay();
glFlush();
}

void Cleanup(void)
{
    DestroyShaders();
    DestroyVB0();
}

int main(int argc, char* argv[])
{
    std::cout << " " << Scor << "\n";
    std::cout << "Player 1 " << Player 2 << "\n";
    std::cout << " 0 " << 0 << "\n";
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(125, 0);
    glutInitWindowSize(1600, 1000);
    glutCreateWindow("Pong");
    glewInit();
    Initialize();
    glutDisplayFunc(RenderFunction);
    glutIdleFunc(invarie);
    glutMouseFunc(mouse);
    glutKeyboardFunc(miscal1);
    glutSpecialFunc(miscal2);
    glutCloseFunc(Cleanup);
    glutMainLoop();
}

```

Pong_Shader.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:
            out_Color = ex_Color;
            break;
        case 1:
            out_Color=vec4 (0.0, 0.0, 0.0, 0.0);
            break;
        case 2:
            out_Color=vec4 (1.0, 1.0, 1.0, 0.0);
            break;
        case 3:
            out_Color=vec4 (0.196078, 0.6, 0.8, 0.0);
            break;
        case 4:
            out_Color=vec4 (0.62352, 0.372549, 0.623529, 0.0);
            break;
        default:
            break;
    };
}
```

Pong_Shader.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;
}
```