**Proiect Grafică**

**Realizat de:**

Bucur Denisa-Andreea

Fănică Narcis-Alexandru

Lăbuș Claudiu-Marian

Micu Diana-Roberta

Nicolae Constantin-Eduard

**Descriere:**

Un prototip de biliard, in care jucatorul controleaza mingea alba, astfel incat sa le loveasca pe cele portocalii.

**Transformari:**

Singurele transformari folosite au fost de tip *Ortho* si *Translate*.  
Translatiile au fost folosite pentru a misca mingea alba, folosind mouse-ul, si pentru a reactiona la coliziuni.

**De ce este original:**

- Metoda de a misca bila alba nu este ca in orice joc de biliard, putem aplica o forta asupra acesteia in orice moment din joc. Aceasta se va misca in directia mouseului, tinand cont si de viteza curenta, si distanta dintre mouse si minge.  
 - Fizicile aplicate mingilor (bug transformat in feature) mingile se pot lipi intre ele in anumite circumstante.

**Cod important:**

I. Cum se misca mingile:

void Ball::Move(float x, float y) {

transform = glm::translate(transform, glm::vec3(x, y, 0));

currentPosition = transform \* initialPosition;

}

void Ball::ApplyForce(glm::vec4 direction, float power) {

glm::vec4 appliedForce = direction \* power;

glm::vec4 frictionForce = -1.0f \* glm::normalize(velocity) \* Physics::FrictionForceValue(mass);

glm::vec4 resultForce = appliedForce + frictionForce;

velocity += resultForce / 100.0f;

CheckForWalls();

Move(velocity[0], velocity[1]);

}

Functia ApplyForce se apeleaza   
 1. La fiecare frame, pt fiecare minge, cu directia ={0,0,0,0}, si power = 0, astfel incat se va aplica doar forța de frecare, daca minga este in miscare.  
 2. Atunci cand apasam click, pt mingea alba, cu directia spre cursor, si power o valoare constanta setata de noi.

II. Coliziunea cu marginile

- Aici ne-a ajutat ChatGPT cu o formula de a calcula directia dupa coliziune.

void Ball::CheckForWalls() {

glm::vec4 wallsNormals[4] = {

{ 1, 0, 0, 0 }, // stanga

{ 0, -1, 0, 0 }, // sus

{ -1, 0, 0, 0 }, // dreapta

{ 0, 1, 0, 0 }, // jos

};

float speedReduction = 1;

if (currentPosition[0] < -0.98f) // stanga

velocity = speedReduction \* (velocity - 2.0f \* glm::dot(velocity, wallsNormals[0]) \* wallsNormals[0]);

// ...

// La fel pt restul peretilor

}

III. Coliziunea dintre bile

- Pentru a calcula velocitatea fiecarei bile dupa coliziune am folosit formulele oferite in documentul 3.

void Billiard::BallsCollision(int ball\_1\_idx, int ball\_2\_idx) {

Ball& ball\_1 = balls[ball\_1\_idx];

Ball& ball\_2 = balls[ball\_2\_idx];

float tempVar = 0.05f;

float xDistance = abs(ball\_1.currentPosition[0] - ball\_2.currentPosition[0]);

float yDistance = abs(ball\_1.currentPosition[1] - ball\_2.currentPosition[1]);

if (xDistance > tempVar / 2 || yDistance > tempVar) return; // too far

glm::vec4 normalVector = ball\_2.currentPosition - ball\_1.currentPosition;

normalVector = glm::normalize(normalVector);

glm::vec4 tangentVector = { -normalVector[1], normalVector[0], 0, 0 };

glm::vec4 v1 = ball\_1.velocity;

glm::vec4 v2 = ball\_2.velocity;

float v1n = glm::dot(normalVector, v1);

float v1t = glm::dot(tangentVector, v1);

float v2n = glm::dot(normalVector, v2);

float v2t = glm::dot(tangentVector, v2);

ball\_1.velocity -= v2n \* normalVector + v2n \* tangentVector;

ball\_2.velocity += v1n \* normalVector + v1n \* tangentVector;

float velocityReducer = 0.7f;

ball\_1.velocity \*= velocityReducer;

ball\_2.velocity \*= velocityReducer;

}

**Bibliografie:**

- 1. <https://www.youtube.com/watch?v=eED4bSkYCB8>

- 2. <https://www.youtube.com/watch?v=nJWtSOxqyLo>

- 3. <https://www.vobarian.com/collisions/2dcollisions2.pdf>

- 4. [https://www.youtube.com/watch?v=FkO6vyMqo8E HYPERLINK "https://www.youtube.com/watch?v=FkO6vyMqo8E&list=PLX2gX-ftPVXVCw9WxxEA4yD14k8yskTSj"& HYPERLINK "https://www.youtube.com/watch?v=FkO6vyMqo8E&list=PLX2gX-ftPVXVCw9WxxEA4yD14k8yskTSj"list=PLX2gX-ftPVXVCw9WxxEA4yD14k8yskTSj](https://www.youtube.com/watch?v=FkO6vyMqo8E&list=PLX2gX-ftPVXVCw9WxxEA4yD14k8yskTSj)  
- 5. ChatGPT