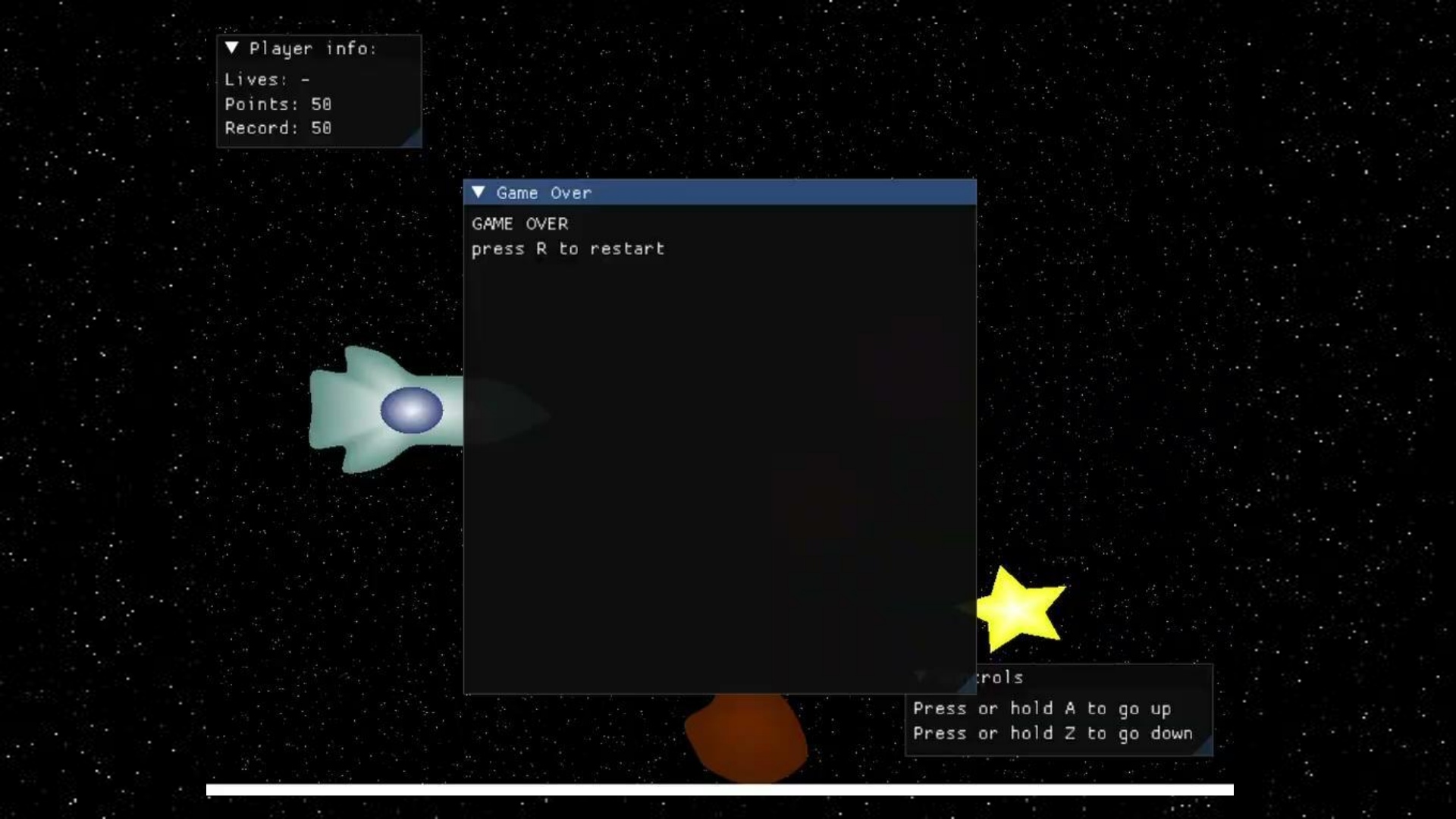
SPACE EXPLORER

Computer Graphics - Elaborato I

di Marilia Merendi


▼ Player info:
Lives: -
Points: 50
Record: 50
▼ Game Over
GAME OVER
press R to restart
rols
Press or hold A to go up
Press or hold Z to go down

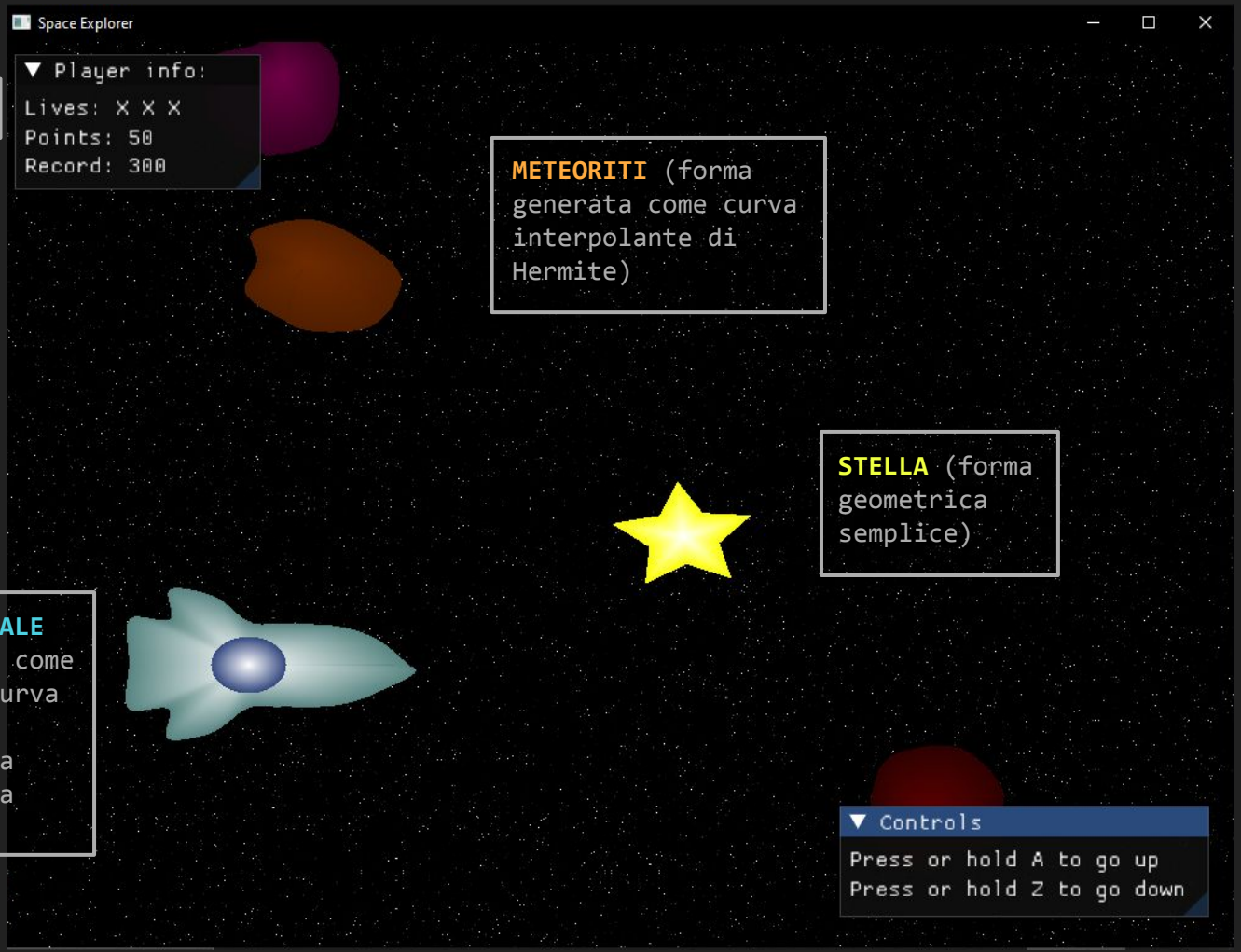
Obiettivi

Il progetto mira a sviluppare un **semplice gioco 2D** a tema spazio, che comprende un giocatore (la navicella spaziale), degli oggetti di scena animati e interattivi (i meteoriti e la stella) e uno sfondo animato (lo spazio).

Il codice ha una struttura principalmente **procedurale**.

Il **main** contiene tutte le principali **inizializzazioni** delle librerie grafiche, le **istanziazioni** degli elementi della scena e un **game loop** in cui viene aggiornato il frame corrente e renderizzata l'immagine.

Finestra ImGui



METEORITI (forma generata come curva interpolante di Hermite)

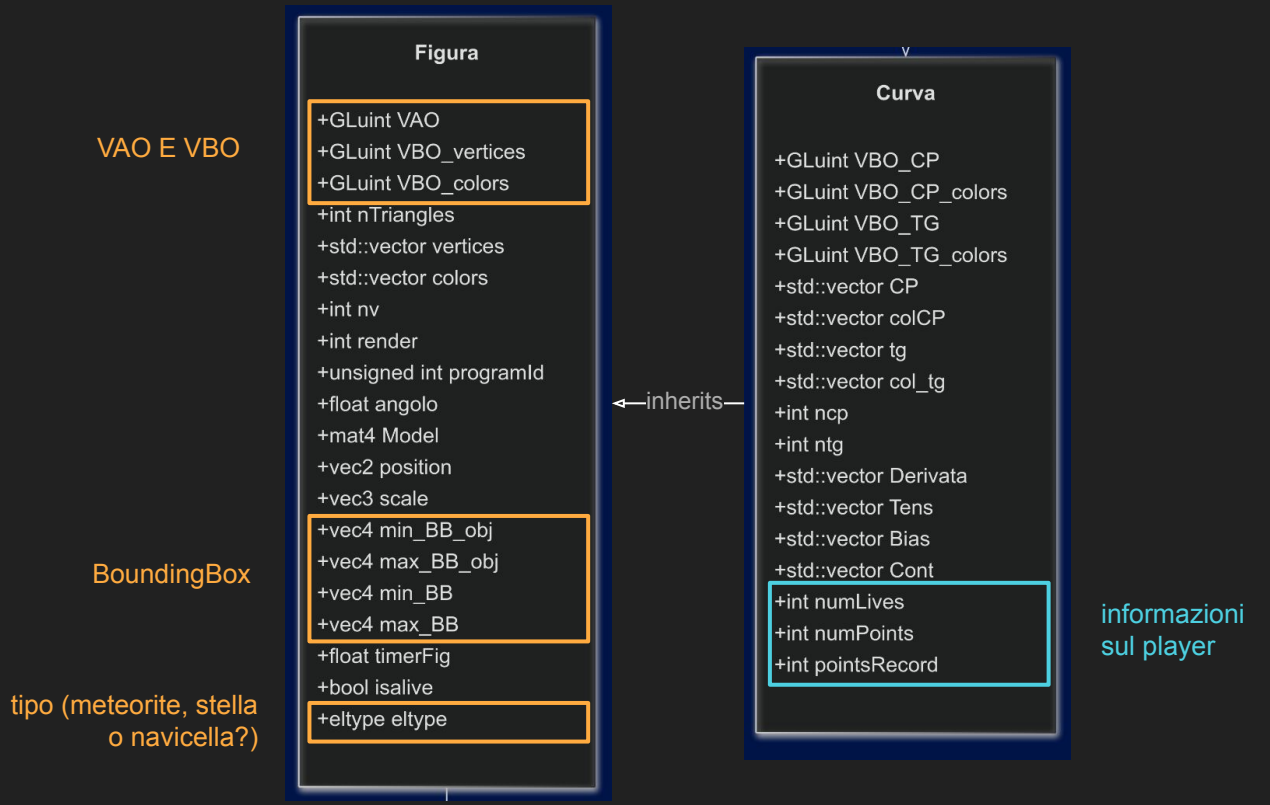
STELLA (forma geometrica semplice)

NAVICELLA SPAZIALE (forma generata come unione di una curva interpolante di Hermite e di una forma geometrica semplice)

Finestra ImGui

Classi ereditarie per gli elementi di gioco (OOP)

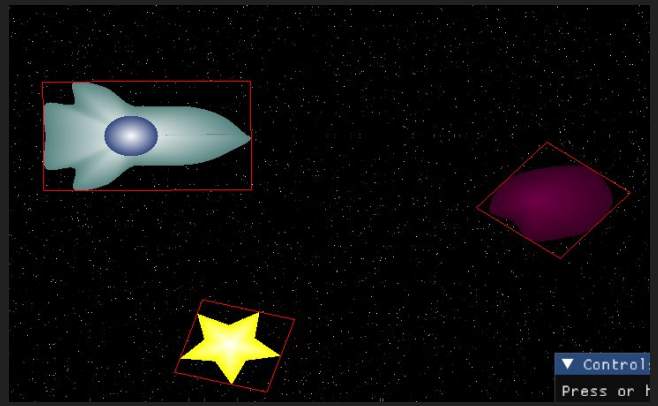
Gli elementi di gioco (navicella, meteoriti e stella) sono stati rappresentati come classi, in particolare per sfruttare un concetto chiave della Programmazione a Oggetti, quello di **ereditarietà**: gli oggetti Curva, dovendo contenere i parametri per l'interpolazione, possono essere visti come delle figure più complesse, e dunque come una specializzazione di Figura.



Gestione delle collisioni

Le collisioni tra la navicella e i meteoriti o la stella sono gestite tramite **Bounding Boxes**.

Una variabile booleana ne permette la visualizzazione in fase di rendering.



init_geometrie.cpp

```
findBB(fig);
fig->vertices.push_back(vec3(fig->min_BB_obj.x, fig->min_BB_obj.y, 0.0));
fig->colors.push_back(vec4(1.0, 0.0, 0.0, 1.0));
fig->vertices.push_back(vec3(fig->max_BB_obj.x, fig->min_BB_obj.y, 0.0));
fig->colors.push_back(vec4(1.0, 0.0, 0.0, 1.0));
fig->vertices.push_back(vec3(fig->max_BB_obj.x, fig->max_BB_obj.y, 0.0));
fig->colors.push_back(vec4(1.0, 0.0, 0.0, 1.0));
fig->vertices.push_back(vec3(fig->min_BB_obj.x, fig->max_BB_obj.y, 0.0));
fig->colors.push_back(vec4(1.0, 0.0, 0.0, 1.0));

fig->nv = fig->vertices.size();
fig->render = GL_TRIANGLE_FAN;
```

I vertici del Bounding Box sono memorizzati in fondo al vettore dei vertici della figura

rendering.cpp

```
//update Bounding Box
updateBB(&player);
updateBB(&Scena[i]);

glUniformMatrix4fv(MatModel, 1, GL_FALSE, value_ptr(Scena[i].Model));
glBindVertexArray(Scena[i].VAO);
//renderizza la shape
glDrawArrays(Scena[i].render, 0, Scena[i].nv - 4);
//renderizza il bounding box (contenuto nelle ultime quattro posizioni dei vertici)
if (show_bounding_boxes)
    glDrawArrays(GL_LINE_LOOP, Scena[i].nv - 4, 4);


//verifica delle collisioni
if (checkCollision(&player, &Scena[i])) {
    Scena[i].isalive = false;
    // Se la figura entrata in collisione e un meteorite, il giocatore perde una vita
    if (Scena[i].eltype == METEORITE && player.isalive == true) {
        player.numLives--;
        if (player.numLives == 0) {
            player.isalive = false;
        }
    }
    //se invece si tratta di una stella, il giocatore guadagna 50 punti
    else if (Scena[i].eltype == STELLA && player.isalive == true) {
        player.numPoints += 50;
    }
}
```

Ad ogni frame, i Bounding Box vengono aggiornati e viene verificato se ci sono state delle collisioni

Creazione della shape Stella

```
// Disegna una stella come funzione matematica basata su un centro e due raggi, uno per le punte e uno per
void INIT_STAR(float cx, float cy, float raggioInterno, float raggioEsterno, Figura* fig)
{
    int t, i;
    float xx, yy;
    float stepA = (2 * PI) / fig->nTriangles;
    fig->vertices.push_back(vec3(cx, cy, 0.0));      //centro
    fig->colors.push_back(vec4(1.0, 1.0, 1.0, 1.0));

    for (i = 0; i <= fig->nTriangles; i++)
    {
        float angle = i * stepA;
        //punte
        xx = raggioEsterno * cos(angle);
        yy = raggioEsterno * sin(angle);

        fig->vertices.push_back(vec3(xx, yy, 0.0));
        fig->colors.push_back(vec4(1.0, 1.0, 0.0, 1.0));

        //indentazioni
        angle += stepA / 2;  //Le punte e le indentazioni si discostano di mezzo angolo l'una dall'altra
        xx = raggioInterno * cos(angle);
        yy = raggioInterno * sin(angle);

        fig->vertices.push_back(vec3(xx, yy, 0.0));
        fig->colors.push_back(vec4(1.0, 1.0, 0.0, 1.0));
    }
```

Fragment shader dello Spazio - utilities

```
#version 330 core

out vec4 fragColor;
in vec2 fragCoord;  // Screen space coordinate (in pixels)
uniform int iFrame;  // The frame count, used to move stars
uniform int height;


// Return random noise in the range [0.0, 1.0], based on a 2D input.
float Noise2d(in vec2 x)
{
    // Simple hash-based noise function
    float xhash = cos(x.x * 37.0);
    float yhash = cos(x.y * 57.0);
    return fract(415.92653 * (xhash + yhash));  // Generate random noise
}

// Generate a starfield based on noise and a threshold for visibility
float StarField(in vec2 vSamplePos, float threshold)
{
    float starVal = Noise2d(vSamplePos);
    if (starVal >= threshold)
    {
        // Apply a sharp cut-off to make stars visible only above a certain threshold
        starVal = pow((starVal - threshold) / (1.0 - threshold), 10.0);  // Sharpen the stars' visibility
    }
    else
    {
        starVal = 0.0;  // No star below threshold
    }
    return starVal;
}
```

Lo shader animato è stato preso da **ShaderToy** e riadattato.

Sfrutta una **noise function** per dare l'effetto di un posizionamento casuale delle stelle e una **soglia di visibilità** per rendere le stelle nitide, come nello spazio.

Fragment shader dello Spazio - main

```
void main()
{
    // Set a background color, fading from dark blue at the top to black at the bottom
    vec3 backgroundColor = vec3(0.0, 0.0, 0.1) * fragCoord.y / height;

    // Define the threshold for star visibility (lower = more stars)
    float threshold = 0.95;  // Adjust this to control star density

    // Control the speed of star movement
    float speed = 0.0005;  // How fast the stars move (higher = faster)

    // Calculate the position of the stars by adding frame-based movement
    vec2 samplePos = fragCoord.xy + vec2(speed * float(iFrame), 0.0);

    // Generate the starfield at the sample position
    float starValue = StarField(samplePos, threshold);

    // Add the stars to the background color
    vec3 starColor = vec3(1.0) * starValue;  // Stars are white

    // Combine the star color with the background
    vec3 finalColor = backgroundColor + starColor;

    // Output the final color
    fragColor = vec4(finalColor, 1.0);
}
```

Utilizza la variabile **iFrame** per scandire il tempo e dare l'effetto dell'animazione.

Grazie per l’attenzione