



Fundamentos de
Computação Gráfica
2025/1



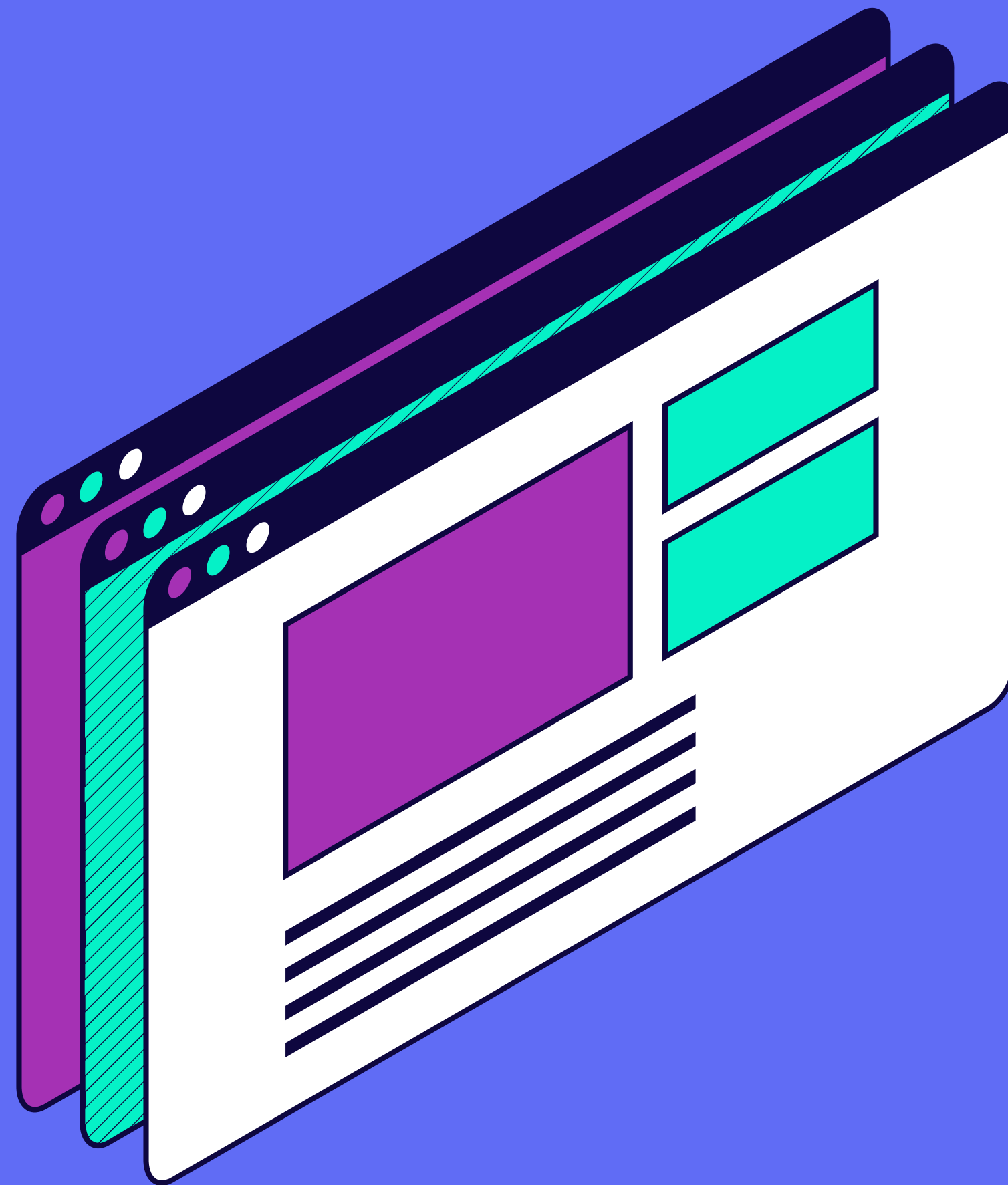
{ Trabalho } Grau A }

Ricardo Moreira Gomes



Índice

- **Apresentação;**
- Demonstração e funcionamento;
- Estrutura geral do código;
- Estrutura dos buffers e shaders;
- Gerenciamento dos sprites;
- Controle do personagem;
- Movimentação de outros objetos;
- Gerenciamento de animações;
- Problemas enfrentados;
- Futuras atualizações.



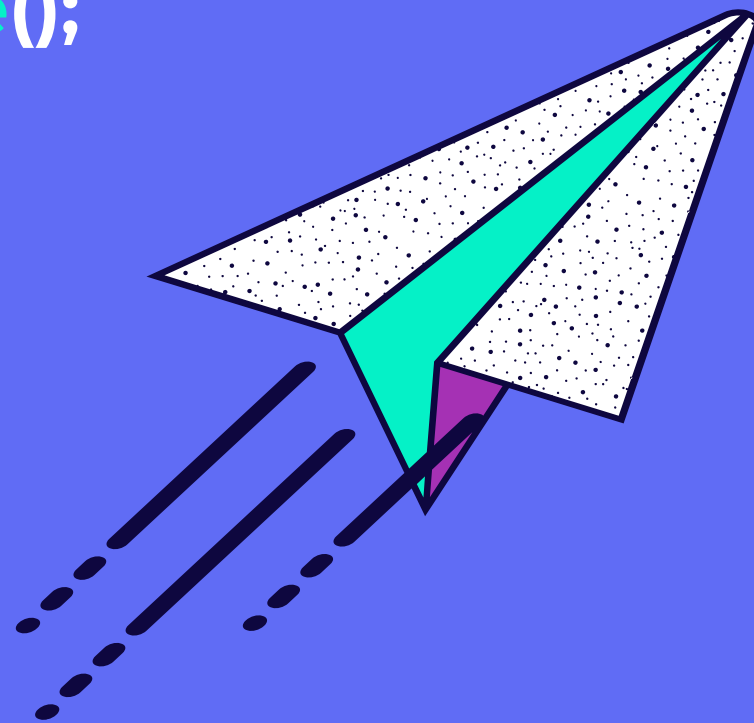
Demonstração & Funcionamento

- Jogo meio **arcade**;
- Pássaro salvando ovos em queda;
- Movimentação nas 4 direções;
- Texturas e spritesheets (**aseprite**);
- Animações (ovos e pássaro);
- Colisões → Pontuação;
- Atualmente **infinito**;
- Melhorias futuras.



Estrutura geral do código

- Relativamente simples;
- **Iteração** sobre o que foi visto em aula;
- Novidades:
 - novos “atributos” em Sprite;
 - nova função **updateSprite()**;
 - vetor **fallingObjects**;
 - colisões, movimentação+;
 - flip no drawSprite().



```
// Inicialização de alguns objetos (ovos) no vetor fallingObjects
for (int i = 0; i < 5; i++)
{
    Sprite obj;
    obj.pos = vec3(0, 0, 0);
    obj.dimensions = vec3(28.0f * 2.0f, 20.0f * 4.0f, 1.0f);
    obj.vel = 2;
    obj.active = false;
    obj.texID = loadTexture("../assets/sprites/ovoroxo2.png");
    obj.VAO = setupSprite(1,2,obj.ds,obj.dt);
    obj.nAnimations = 1;
    obj.nFrames = 2;
    obj.angle = 0.0;
    obj.iAnimation = 0;
    obj.iFrame = 0;
    obj.animTimer = 0.0f;
    fallingObjects.push_back(obj); // DEIXAR POR ÚLTIMO
}
```

```
// Movimento e colisão dos objetos
for (auto& obj : fallingObjects)
{
    if (obj.active)
    {
        obj.pos.y -= obj.vel; /* lastTime; (velocidade só cresce)

        // Verifica colisão com o jogador (spr1)
        if (obj.pos.y <= spr1.pos.y + spr1.dimensions.y/2 &&
            obj.pos.y >= spr1.pos.y - spr1.dimensions.y/2 &&
            obj.pos.x >= spr1.pos.x - spr1.dimensions.x/2 &&
            obj.pos.x <= spr1.pos.x + spr1.dimensions.x/2)
        {
            obj.active = false; // Apaga o objeto
            score++; // Aumento de pontuação

            if(spr1.vel <= 1); // Diminuição de velocidade até 1 (mínimo)
            else
            {
                spr1.vel -= 0.2f;
            }
        }

        // Verifica se saiu da tela -> apaga objeto
        if (obj.pos.y < -50.0f)
        {
            obj.active = false;
        }
    }
}
```

```
void updateSprite(Sprite& sprite, float deltaTime) {
    if(sprite.nFrames <= 1) return; // Sem animação

    sprite.animTimer += deltaTime;
    float frameDuration = 1.0f / FPS; // Sincronia de fluidez com FPS

    if(sprite.animTimer >= frameDuration) {
        sprite.animTimer = 0.0f;
        sprite.iFrame = (sprite.iFrame + 1) % sprite.nFrames; // Avanço de frames
    }
}
```

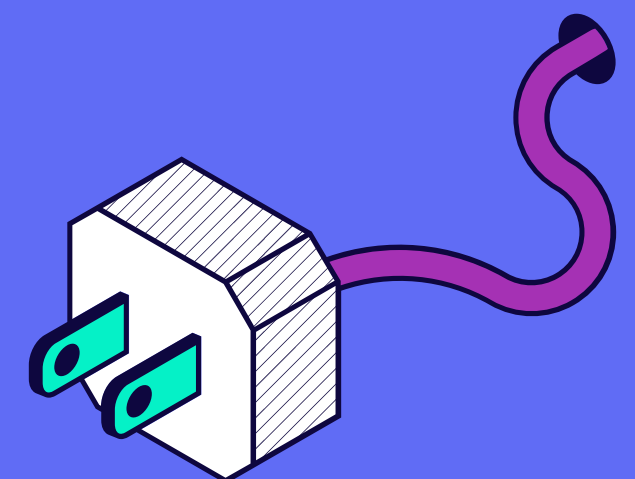
```
// Flip horizontal (movimento alinhado a direção do pássaro)
if(spr.flipped)
{
    model = scale(model, vec3(-spr.dimensions.x, spr.dimensions.y, 1.0f));
}
else
{
    model = scale(model, spr.dimensions);
}
```

Estrutura dos buffers e shaders

- Pipeline da OpenGL moderna desenvolvido em aula;
- `setupSprite()` → configura os buffers dos **vértices**;
 - animação, tamanhos, vértices (e seus atributos);
 - criação de **VBO/VAO**.
- Vertex shader + Fragment shader;
- Ortografia fixa + `main()`:
 - `setupShader()` + `setupSprite()` + `loadTexture()`;
 - loop de renderização + `glDrawArrays()`;
 - matriz → `drawSprite()`
 - animações via `updateSprite` (offset, ds/dt etc.);

```
void setupSprite() {
    // Definindo o VBO (Vertex Buffer Object)
    GLuint VBO;
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // Definindo o VAO (Vertex Array Object)
    GLuint VAO;
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    // Definindo os vértices
    float vertices[] = {
        // Posição (x, y, z)
        // Textura (u, v)
        // Cor (r, g, b)
        // Opacidade (a)
        -1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        -1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        -1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        -1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f
    };
    // Definindo os índices
    GLuint indices[] = {
        0, 1, 2, 3, 3, 2, 2, 1, 1, 0,
        4, 5, 6, 7, 7, 6, 6, 5, 5, 4
    };
    // Definindo o tamanho dos buffers
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
    // Definindo os atributos
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(5 * sizeof(float)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(3, 1, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(7 * sizeof(float)));
    glEnableVertexAttribArray(3);
    // Definindo o tamanho dos índices
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
}
```

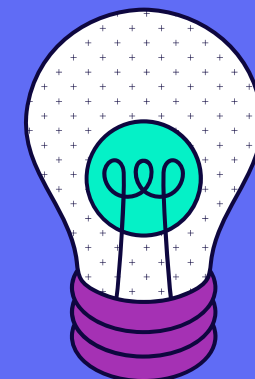
```
void setupShader() {
    // Definindo o Vertex Shader
    const char* vertexShaderSource = "#version 330\nin vec3 pos;in vec2 tex;in vec3 col;out vec4 fragColor;void main(){fragColor = vec4(col, 1.0);}";
    GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    // Definindo o Fragment Shader
    const char* fragmentShaderSource = "#version 330\nout vec4 fragColor;void main(){fragColor = vec4(1.0, 1.0, 1.0, 1.0);}";
    GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShader);
    // Definindo o Programa
    GLuint program = glCreateProgram();
    glAttachShader(program, vertexShader);
    glAttachShader(program, fragmentShader);
    glLinkProgram(program);
    // Definindo os atributos
    GLuint VBO;
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // Definindo o VAO
    GLuint VAO;
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    // Definindo os vértices
    float vertices[] = {
        // Posição (x, y, z)
        // Textura (u, v)
        // Cor (r, g, b)
        // Opacidade (a)
        -1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        -1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        -1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
        -1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f
    };
    // Definindo os índices
    GLuint indices[] = {
        0, 1, 2, 3, 3, 2, 2, 1, 1, 0,
        4, 5, 6, 7, 7, 6, 6, 5, 5, 4
    };
    // Definindo o tamanho dos buffers
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
    // Definindo os atributos
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(5 * sizeof(float)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(3, 1, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(7 * sizeof(float)));
    glEnableVertexAttribArray(3);
    // Definindo o tamanho dos índices
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
}
```



Gerenciamento dos Sprites

- Struct **Sprite** desenvolvido em aula:
 - bool active, bool flipped, float animTimer;
- 2 triângulos no VAO (criado uma vez para cada sprite);
- Texturas são carregadas via loadTexture();
- drawSprite() faz a renderização (vincula VAO etc.);
- updateSprite() faz o controle de animação via deltaTime;
- Vetor fallingObjets faz o gerenciamento dinâmico dos objetos em queda (ovos);
 - Reutilização via objetos ativos **x** inativos;

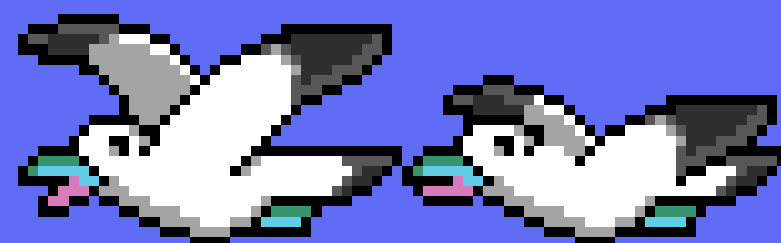
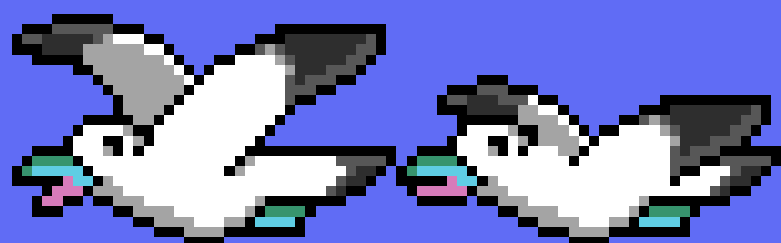
```
struct Sprite
{
    GLuint VAO;
    GLuint texID;
    vec3 pos;
    vec3 dimensions;
    float angle;
    float vel;
    int nAnimations, nFrames;
    int iFrame, iAnimation;
    float ds, dt;
    bool active;           // Para objetos ativos
    bool flipped;         // Para flipar horizontalmente
    float animTimer;      // Timer para animação
};
```



Controle do personagem

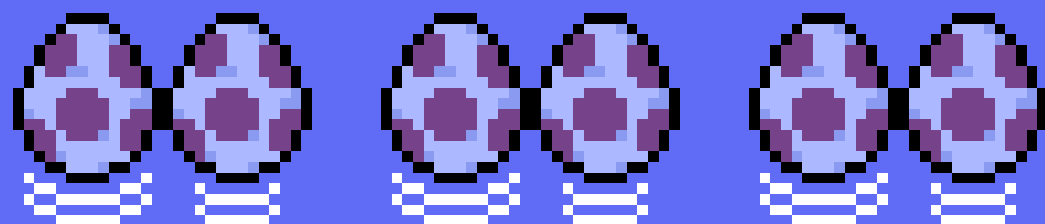
- `glfwPollEvents()` → funções de callback
- Movimentação nas 4 direções:
 - cima/baixo, esquerda/direita
- Velocidade fixa (`spr1.vel = 4.0`);
- Velocidade diminui 0.2 a cada ovo salvo;
 - Até chegar em 1.0 (velocidade mínima).

```
// Detecção dos inputs
if (keys[GLFW_KEY_LEFT] == true || keys[GLFW_KEY_A] == true) // Esquerda
{
    spr1.pos.x -= spr1.vel;
    spr1.flipped = false;
}
if (keys[GLFW_KEY_RIGHT] == true || keys[GLFW_KEY_D] == true) // Direita
{
    spr1.pos.x += spr1.vel;
    spr1.flipped = true;
}
if (keys[GLFW_KEY_UP] == true || keys[GLFW_KEY_W] == true) // Cima
{
    spr1.pos.y += spr1.vel;
}
if (keys[GLFW_KEY_DOWN] == true || keys[GLFW_KEY_S] == true) // Baixo
{
    spr1.pos.y -= spr1.vel;
}
```



Movimentação de outros objetos

- Inicialização de alguns Sprite obj **ATIVOS**;
- **push_back** → vetor fallingObjects;
- timers para spawn de objetos **INATIVOS**;
 - inativos = colidiram ou estão fora da tela;
- Instancia em posição aleatória no topo da tela;
- Colisão com personagem → INATIVO;
- Saiu da tela → **INATIVO**;
- Velocidade fixa **obj.vel == 2**;



```
// Spawn de objetos
spawnTimer += deltaTime;
if (spawnTimer >= spawnInterval)
{
    spawnTimer = 0.0f;
    for (auto& obj : fallingObjects) // Encontra um objeto INATIVO para reutilizar
    {
        if (!obj.active)
        {
            obj.pos.x = rand() % 700 + 50; // Posição x aleatória
            obj.pos.y = 600.0f; // Começa no topo
            obj.active = true; // Volta a ser ativo
            break;
        }
    }
}
```

```
// Inicialização de alguns objetos (ovos) no vetor fallingObjects
for (int i = 0; i < 5; i++)
{
    Sprite obj;
    obj.pos = vec3(0, 0, 0);
    obj.dimensions = vec3(28.0f * 2.0f, 20.0f * 4.0f, 1.0f);
    obj.vel = 2;
    obj.active = false;
    obj.texID = loadTexture("../assets/sprites/ovoroxo2.png");
    obj.VAO = setupSprite(1,2,obj.ds,obj.dt);
    obj.nAnimations = 1;
    obj.nFrames = 2;
    obj.angle = 0.0;
    obj.iAnimation = 0;
    obj.iFrame = 0;
    obj.animTimer = 0.0f;
    fallingObjects.push_back(obj); // DEIXAR POR ÚLTIMO
}
```


Gerenciamento de animações

- struct **Sprite** → nAnimations, nFrames, iFrame, iAnimation, ds, dt etc;
 - animTimer (timer interno entre frames);
- Background fixo, pássaro/ovos com **2** frames;
- Generalização via **updateSprite()**;
- Utilização de deltaTime por razões de FPS;
- Loop de frames;
- load de spritesheet → main() → updateSprite()

```
// Render/process dos objetos ativos no vetor fallingObjects
for(auto& objeto : fallingObjects)
{
    if (objeto.active) // Não coletado nem fora da tela
    {
        // Offset de animação de obj (ovos)
        float offsetSovo = objeto.iFrame * objeto.ds;
        float offsetTovo = objeto.iAnimation * objeto.dt;
        glUniform2f(glGetUniformLocation(shaderID, "offset_tex"), offsetSovo, offsetTovo);
        drawSprite(shaderID, objeto);
        updateSprite(objeto, deltaTime);
    }
}

// Offset de animação de spr1 (pássaro)
float offsetS = spr1.iFrame * spr1.ds;
float offsetT = spr1.iAnimation * spr1.dt;
glUniform2f(glGetUniformLocation(shaderID, "offset_tex"), offsetS, offsetT);
drawSprite(shaderID, spr1);
updateSprite(spr1, deltaTime);
```

```
// Função simples para lidar com animações
// Atualiza iFrame com base no tempo passado
void updateSprite(Sprite& sprite, float deltaTime) {
    if(sprite.nFrames <= 1) return; // Sem animação

    sprite.animTimer += deltaTime;
    float frameDuration = 1.0f / FPS; // Sincronia de fluidez com FPS

    if(sprite.animTimer >= frameDuration) {
        sprite.animTimer = 0.0f;
        sprite.iFrame = (sprite.iFrame + 1) % sprite.nFrames; // Avanço de frames
    }
}
```



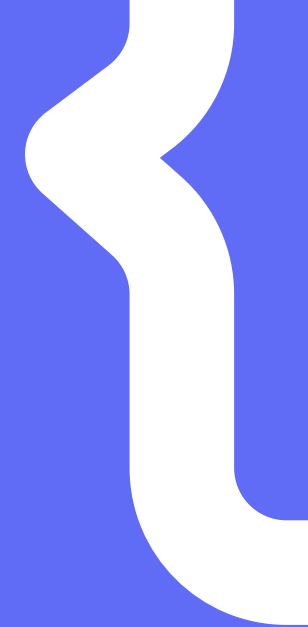
Problemas Enfrentados e Futuras Atualizações

- Acúmulo de variáveis para timers/delta → `updateSprite()`;
- Wrap de spritesheet acaba errado → correção 50%;
- Velocidade de queda `infinita` nos ovos ao multiplicar por `lastTime`;
- Precisão nos `.dimensions` nos carregamentos;
- Ajustes de `.vel` tanto para `obj` quanto para `sprl`;
- Tempo para incluir melhorias/finalizações:
 - `Parallax` com 3 camadas;
 - Pontuação via `HUD` (FreeType vai dar algum trabalho);
 - `GameOver` (quando tempo acabar, mostra pontuação conforme quantidade de ovos coletados).





<https://github.com/rikmgomes/fundcompgraf202501>



Obrigado pela atenção!

Ricardo
Moreira
Gomes

{ Trabalho
Grau A }

Fundamentos de
Computação
Gráfica 2025/1

