

Computação Gráfica – Trabalho 3

JOGO DE DAMAS

- Prof.: Marcelo Gattass
- Nome: Luiz Felipe Machado da Silva

Computação Gráfica – Trabalho 3

■ Desenho do tabuleiro

```
glEnable(GL_LIGHTING);

glPushMatrix();
{
    // translada para o centro da coordenadas
    glTranslatef(-22.5 * 2.0f, -22.5 * 2.0f, 0.0f);

    bool alterna_tex = false;
    for(int i = 0; i < 10; i++)
    {
        for(int j = 0; j < 10; j++)
        {
            app3->load(cAppearance::TEXTURE, 1.0);

            if(i == 0 || j == 0 || i == 9 || j == 9) tex3->load();
            else
            {
                glPushName(20); glPushName(i); glPushName(j);

                if(i == m_focus_linha + 1 && j == m_focus_coluna + 1)
                {
                    glPushMatrix();
                    {
                        glDisable(GL_LIGHTING);
                        {
                            glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
                            glScalef(5.0f, 5.0f, 1.0f);
                            glTranslatef(i * 2.0f, j * 2.0f, 0.10f);
                            c.desenha_linhas_superiores();
                        }
                        glEnable(GL_LIGHTING);
                    }
                    glPopMatrix();
                }

                alterna_tex ? tex1->load() : tex2->load();
            }
        }
    }

    glPushMatrix();
    {
        glScalef(5.0f, 5.0f, 1.0f);
        glTranslatef(i * 2.0f, j * 2.0f, 0.0f);
        c.desenha();
    }
    glPopMatrix();

    if(i == 0 || j == 0 || i == 9 || j == 9) tex1->unload();
    else
    {
        alterna_tex ? tex1->unload() : tex2->unload();
        glPopName(); glPopName(); glPopName();
    }

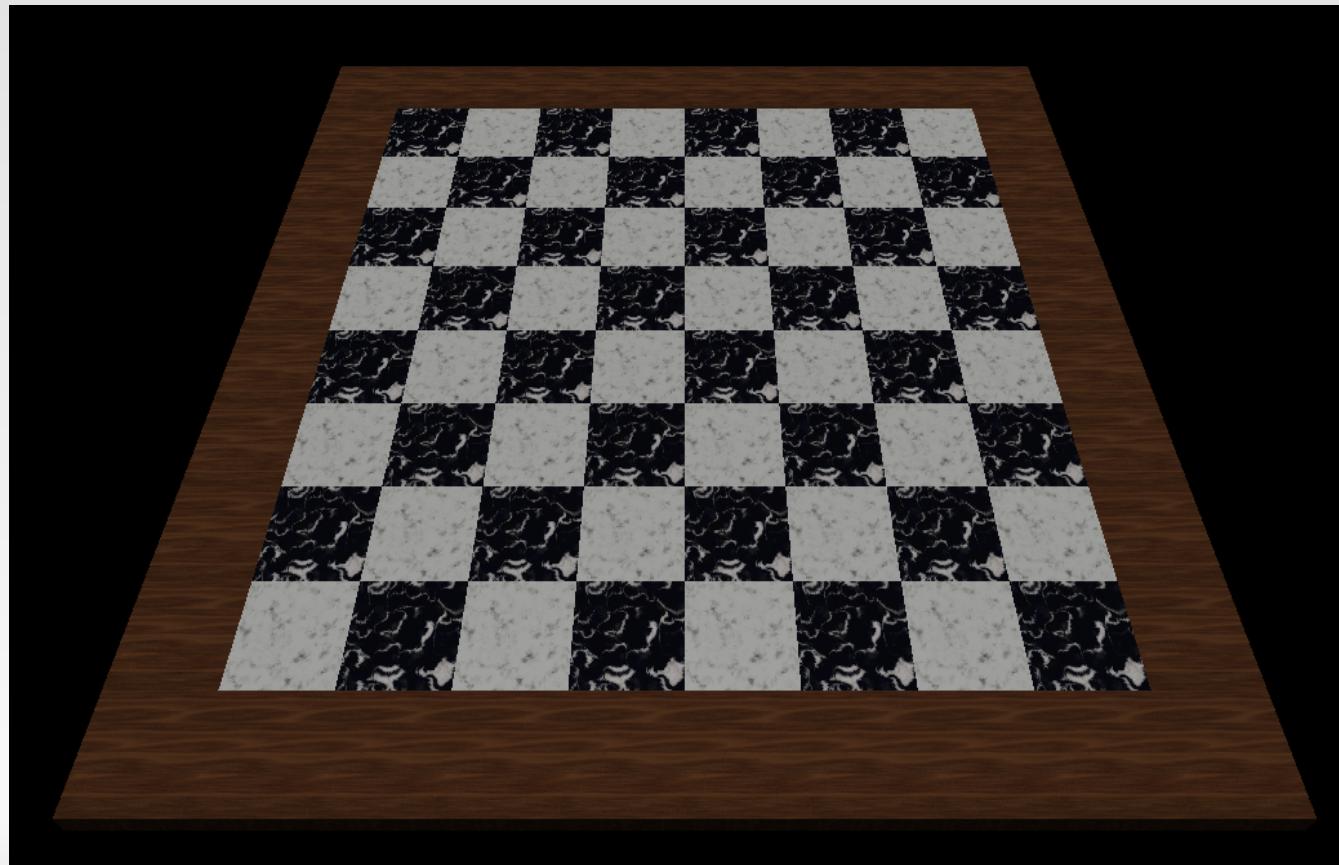
    alterna_tex = !alterna_tex;
}

alterna_tex = !alterna_tex;

}
glPopMatrix();
```

Computação Gráfica – Trabalho 3

- Desenho do tabuleiro



Computação Gráfica – Trabalho 3

- Estrutura do jogo:
 - `vector< vector< cPeca* > > m_game;`

Computação Gráfica – Trabalho 3

- Estrutura da peça:

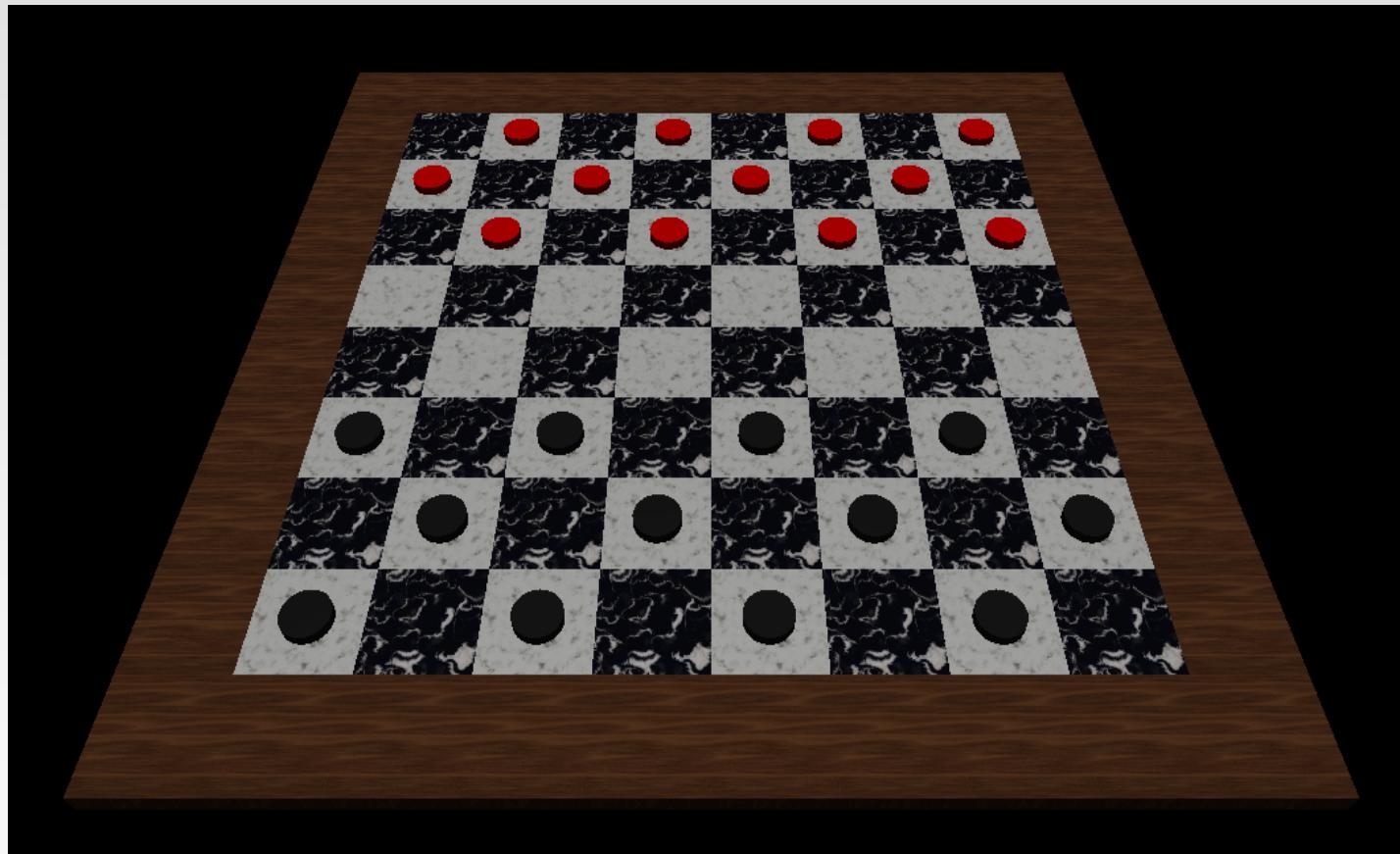
```
private:  
    bool m_is_focus;  
    string m_jogador;  
    cCoord<float> m_t;  
    cAppearance *m_app;
```

- Desenho da peça:

```
if(is_focus)  
    m_app->load(cAppearance::PICK);  
else  
    m_app->load(cAppearance::COLOR);  
  
glPushName((int)10); // push peça id  
glPushName((int)m_t.x); // push linha  
glPushName((int)m_t.y); // push coluna  
  
glPushMatrix();  
{  
    glTranslatef(m_t.x * 10.0f, m_t.y * 10.0f, m_t.z);  
    glScalef(2.3, 2.3, 1);  
    gluCylinder(quadric, 1.0f, 1.0f, 1.0f, 32, 32);  
    gluDisk(quadric, 0.0f, 1.0f, 32, 32);  
    glTranslatef(0.0f, 0.0f, 1.0f);  
    gluDisk(quadric, 0.0f, 1.0f, 32, 32);  
}  
glPopMatrix();  
  
glPopName();  
glPopName();  
glPopName();
```

Computação Gráfica – Trabalho 3

- Desenho da Peça:



Computação Gráfica – Trabalho 3

- Seleção:

```
// realiza o pick
int x_selected = x;
int y_selected = h_b - y;

GLuint buff[64] = {0};
GLint hits, view[4];

// buffer onde vai receber os valores de selecao
glSelectBuffer(64, buff);

// guarda a view
glGetIntegerv(GL_VIEWPORT, view);

// ativa o modo de selecao
glRenderMode(GL_SELECT);

// limpa os nomes anteriormente armazenados
glInitNames();

// coloca um elemento na pilha de nomes
glPushName(0);

// modifica a visualizacao de acordo com o cursor
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();

// restringe o desenho para um area em volta do cursor
gluPickMatrix(x_selected, y_selected, 1.0, 1.0, view);
gluPerspective(50, (float)view[2] / (float)view[3], 0.0001, 500.0);
```

```
// desenha na tela
glMatrixMode(GL_MODELVIEW);

// desenha novamente e preeche o array de nomes
render();

glMatrixMode(GL_PROJECTION);
glPopMatrix();

// retorna a quantidade de objetos selecionados
hits = glRenderMode(GL_RENDER);

int l, c;

// busca linha e coluna selecionado
l = (int)buff[5];
c = (int)buff[6];

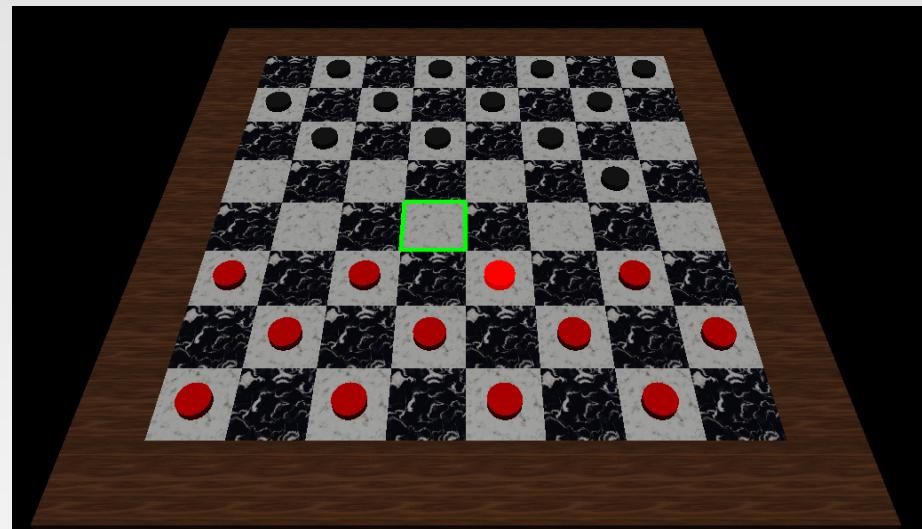
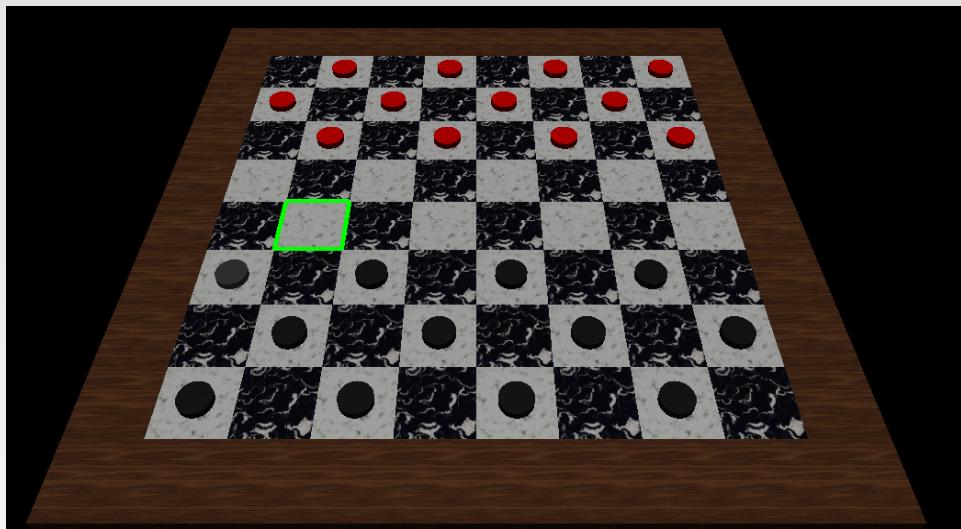
// verifica se eh valida a linha e a coluna selecionada
if(l > 0 && c > 0 && l <= 8 && c <= 8)
{
    if((int)buff[11] == 10)
    {
        // foi selecionada uma peça
        de_linha = l;
        de_coluna = c;
    }
    else if((int)buff[4] == 20)
    {
        // foi selecionado um quadrado (espaco) do tabuleiro
        para_linha = l;
        para_coluna = c;
    }

    // atualiza foco
    set_focus();
    glutPostRedisplay();
}

glMatrixMode(GL_MODELVIEW);
```

Computação Gráfica – Trabalho 3

- Seleção:



Computação Gráfica – Trabalho 3

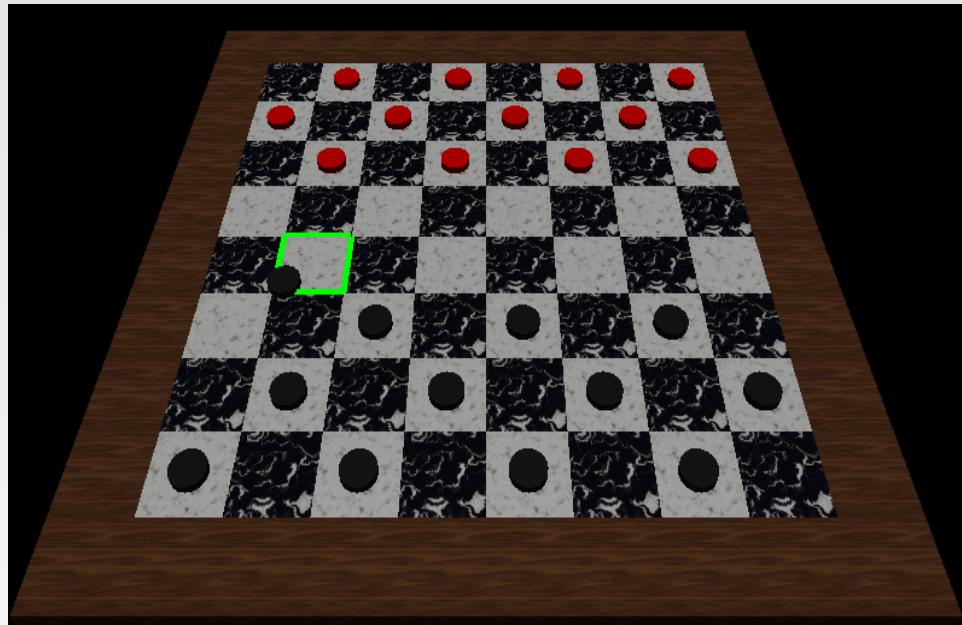
- Animação:

```
void glutTimerFunc ( unsigned int msecs , void (*func)(int value), value );
```

- msecs - Number of milliseconds to pass before calling the callback.
- func - The timer callback function.
- value - Integer value to pass to the timer callback.

Computação Gráfica – Trabalho 3

- Animação: Peça andando



```
// animacao da peça andando 1 casa
float tx, ty, tz;

// busca a peça que ja se movimentou
cPeca *p = game->get_peca(para_linha, para_coluna);
p->get_t(tx, ty, tz);

if(fabs(((float)tx - para_linha)) < step_peca &&
   fabs(((float)ty - para_coluna)) < step_peca)
{
    // chegou no destino // para de animar
    estah_animando = false;
    p->set_t(para_linha, para_coluna, 1.0f);
    de_linha = de_coluna = para_linha = para_coluna = -1;
    set_focus();
    espera = true;
}
else
{
    // ainda nao chegou no destino

    // verifica se soma ou subtrai
    int a = 1;
    if(para_linha - de_linha < 0)
        a = -1;
    tx = tx + (a * step_peca);

    a = 1;
    if(para_coluna - de_coluna < 0)
        a = -1;
    ty = ty + (a * step_peca);

    p->set_t(tx, ty, 1.01f);
    glutTimerFunc(tempo_troca, cb_animacao, 0);
}
```

Computação Gráfica – Trabalho 3

- Animação: Peça "comendo"

```
float tx, ty, tz;
// busca a peça que já se movimentou
cPeca *p = game->get_peca(para_linha, para_coluna);
p->get_t(tx, ty, tz);

if(fabs((float)tx - para_linha) < step_peca &&
    fabs((float)ty - para_coluna) < step_peca)
{
    // chegou no destino
    // para de animar
    estah_animando = false;
    p->set_t(para_linha, para_coluna, 1.0f);
    de_linha = de_coluna = para_linha = para_coluna = -1;
    set_focus();
    espera = true;
}
else
{
    // verifica se está passando por cima da peça que vai ser comida
    int come_linha = (de_linha + para_linha) / 2;
    int come_coluna = (de_coluna + para_coluna) / 2;
    float come_tx, come_ty, come_tz;
    cPeca *come_p = game->get_peca(come_linha, come_coluna);
    if(come_p)
    {
        come_p->get_t(come_tx, come_ty, come_tz);
        if(fabs((float)tx - come_tx) < step_peca &&
            fabs((float)ty - come_ty) < step_peca)
        {
            // estah em cima da peça que vai ser comida
            // remove a peça
            delete come_p;
            game->set_peca(NULL, come_linha, come_coluna);
        }
    }
}
```

```
// ainda não chegou no destino
// verifica se soma ou subtrai
int a = 1;
if(para_linha - de_linha < 0)
    a = -1;
tx = tx + (a * step_peca);

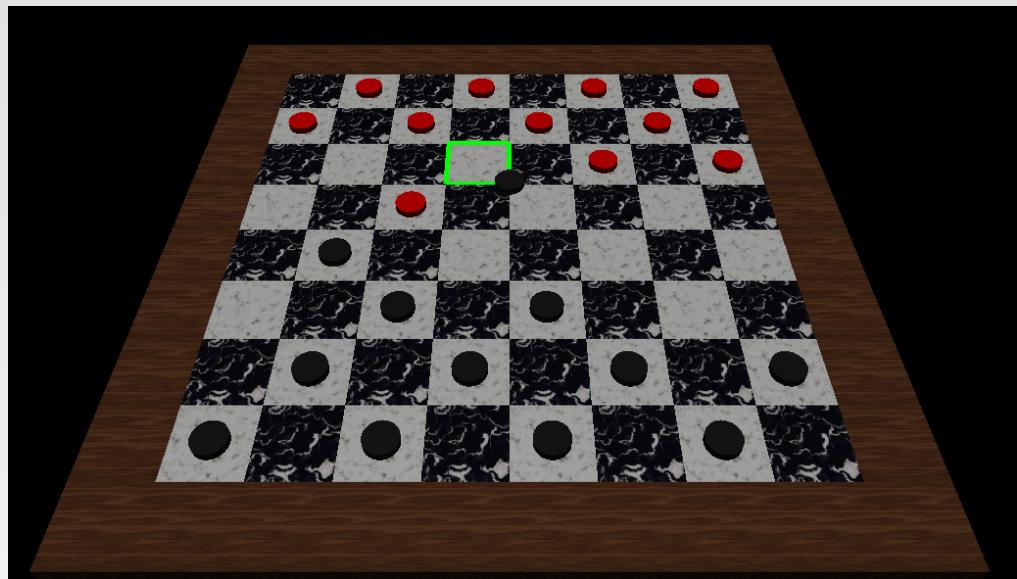
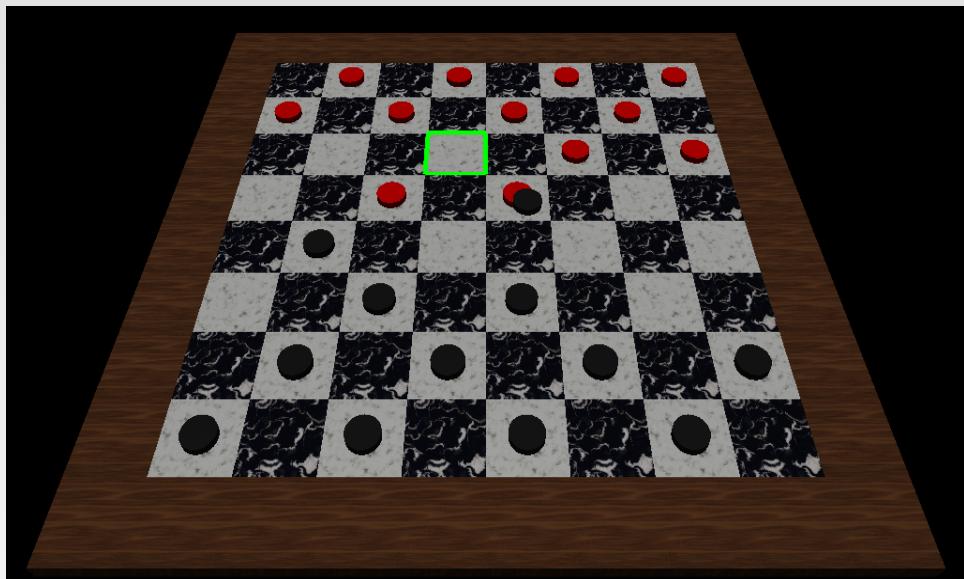
a = 1;
if(para_coluna - de_coluna < 0)
    a = -1;
ty = ty + (a * step_peca);

p->set_t(tx, ty, 1.0f);

glutTimerFunc(tempo_troca, cb_animacao, 1);
}
```

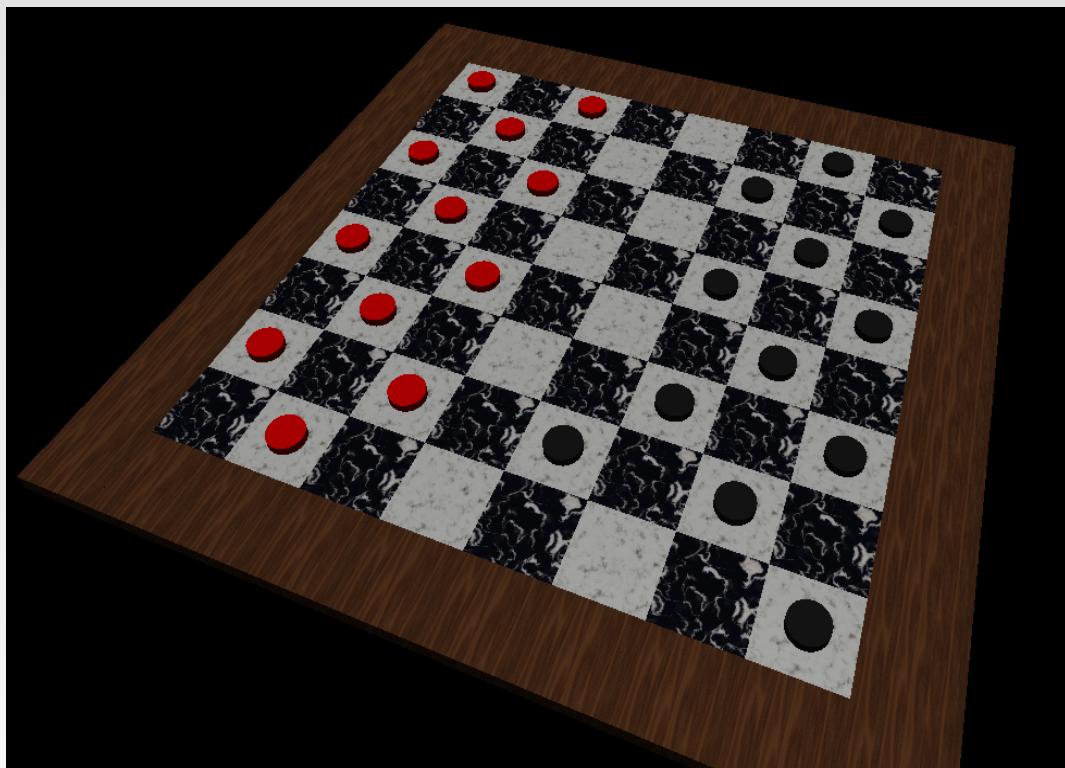
Computação Gráfica – Trabalho 3

- Animação: Peça "comendo"



Computação Gráfica – Trabalho 3

- Animação: Troca de jogador



```
else if(i == 6)
{
    // incrementa o angulo
    angle += step_camera;

    if(vez_jogador1)
    {
        // vez do jogador 1
        // rotaciona ate h chegar a 180
        if(angle < 180)
            glutTimerFunc(tempo_troca, cb_animacao, i);
        else// chegou a 180 graus
            vez_jogador1 = !vez_jogador1;
    }
    else
    {
        // vez do jogador 2
        // rotaciona ate h terminar de dar a volta
        if(angle < 360)
            glutTimerFunc(tempo_troca, cb_animacao, i);
        else
        {
            // de uma volta completa (360)
            angle = 0.0f; // reinicia
            vez_jogador1 = !vez_jogador1;
        }
    }
}
```

Computação Gráfica – Trabalho 3

- Mensagem de erro:

```
void desenha_texto(char *t, float x, float y, float z)
{
    glDisable(GL_LIGHTING);
    {
        glColor4f(1 - cor_fundo.x, 1 - cor_fundo.y, 1 - cor_fundo.z, 1.0f);
        glPushMatrix();
        {
            glRasterPos3f(x, y, z);
            while(*t) glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *t++);
        }
        glPopMatrix();
    }
    glEnable(GL_LIGHTING);
}
```

