



UNIVERSIDADE FEDERAL DE RORAIMA CENTRO DE CIÊNCIA E
TECNOLOGIA BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DCC703 – COMPUTAÇÃO GRÁFICA (2024.2)

Data de entrega: 19/03/2025

DISCENTES:

FELIPE RUBENS DE SOUSA BORGES (2020020120)

MATEUS MORAES DE MOURA (2019027100)

COMPUTAÇÃO GRÁFICA

Relatório do Projeto Final – Jogo Tetris

**Boa Vista-RR
2024.2**

Relatório do Projeto Final – Jogo Tetris

Relatório apresentado para o projeto final da disciplina de Computação Gráfica, ofertada pelo curso de Ciência da Computação da Universidade Federal de Roraima.

Prof. Luciano Ferreira

Sumário

COMPUTAÇÃO GRÁFICA	1
Relatório do Projeto Final – Jogo Tetris	2
Introdução	4
Roteiro do Jogo.....	4
Ferramentas Usadas	7
Principais Técnicas de Computação Gráfica	7
Telas do Jogo	11
Conclusões.....	11

Introdução

Este relatório descreve a implementação do projeto "TETRIS com Melhorias Visuais" que é uma reinterpretação do clássico jogo Tetris, desenvolvido com a biblioteca Pygame em Python. Este trabalho visa não apenas recriar a jogabilidade tradicional, mas também enriquecer a experiência do usuário com elementos visuais avançados, como gradientes, sombras, animações e partículas. O objetivo é demonstrar a aplicação de técnicas de Computação Gráfica para tornar o jogo mais atraente e funcional, mantendo a essência do Tetris original enquanto se adicionam melhorias como menu interativo, sistema de pontuação com recordes, níveis de dificuldade e efeitos sonoros.

Roteiro do Jogo

O jogo segue uma estrutura cíclica com as seguintes etapas principais:

1. **Menu Inicial:** O jogador é apresentado a um menu com opções para jogar, visualizar recordes, ajustar a dificuldade ou sair.
2. **Seleção de Dificuldade:** Caso escolhida, permite ao jogador optar entre fácil, médio ou difícil, alterando a velocidade de queda das peças.
3. **Jogabilidade:** O jogador controla peças (tetrominós) que caem, podendo movê-las lateralmente, rotacioná-las ou acelerar sua descida, enquanto tenta completar linhas para ganhar pontos.
4. **Game Over:** Ao preencher a grade até o topo, o jogo termina, solicitando o nome do jogador para salvar a pontuação e o tempo no ranking de recordes.
5. **Retorno ao Menu:** Após o fim da partida, o jogador retorna ao menu principal para novas ações.

Detalhes do Jogo

Detalhes Gerais:

- **Tela:** Resolução de 600x800 pixels, com uma grade de 10x20 blocos (cada bloco com 30x30 pixels).
- **Peças:** Sete tetrominós clássicos (I, O, T, S, Z, L, J), cada um com uma cor distinta.
- **Controles:** Setas direcionais (esquerda, direita, baixo), tecla "cima" para rotação, "espaço" para queda instantânea, "P" para pausar e "Q" para sair da fase.
- **Pontuação:** 100 pontos por linha completada, com registro de linhas concluídas e tempo decorrido.
- **Dificuldade:** Três níveis (fácil: velocidade 1, médio: 2, difícil: 3), ajustáveis no menu.
- **Efeitos Visuais:** Inclui gradiente no fundo, sombra da peça, partículas ao fixar peças, animação de piscar ao remover linhas e texto 3D.
- **Áudio:** Música tema do Tetris em loop com volume ajustado a 10%.

Detalhes das Cores definidas no Código:

1. Cores Fixas

Essas cores são usadas para elementos específicos e não variam durante o jogo:

- **BLACK = (0, 0, 0)**
 - Descrição: Preto puro (sem luz em nenhum canal RGB).
 - Uso:
 - Fundo padrão da tela antes de aplicar o gradiente (`screen.fill(BLACK)`).
 - Cor base para limpar a tela em várias funções, como `show_menu()` e `get_player_name()`.
 - Finalidade: Garante um fundo neutro que não interfere com outros elementos visuais.
- **WHITE = (255, 255, 255)**
 - Descrição: Branco puro (máxima intensidade em todos os canais RGB).
 - Uso:
 - Cor padrão para textos (ex.: pontuação, menu, recordes).
 - Usada na animação de piscar ao remover linhas completas (`remove_complete_lines()`).
 - Finalidade: Alta visibilidade contra o fundo escuro e feedback visual claro em animações.
- **GRAY = (50, 50, 50)**
 - Descrição: Cinza escuro (baixa intensidade uniforme nos canais RGB).
 - Uso:
 - Bordas da grade quando uma célula está vazia (`draw_grid()` desenha retângulos com espessura 1).
 - Finalidade: Delimita visualmente as células da grade sem distrair das peças coloridas.

2. Cores das Peças (COLORS)

O array **COLORS** contém as cores usadas para as sete peças do Tetris (tetromínos). Cada peça recebe uma cor distinta para facilitar a identificação:

- **COLORS = [(0, 255, 255), (255, 255, 0), (255, 0, 255), (0, 255, 0), (255, 0, 0), (255, 165, 0), (0, 0, 255)]**
 - Detalhes:
 1. (0, 255, 255) - Ciano: Máximo de verde e azul, sem vermelho. Usada também em textos 3D no menu.
 2. (255, 255, 0) - Amarelo: Máximo de vermelho e verde, sem azul.

3. (255, 0, 255) - Magenta: Máximo de vermelho e azul, sem verde.
 4. (0, 255, 0) - Verde: Máximo de verde, sem vermelho ou azul.
 5. (255, 0, 0) - Vermelho: Máximo de vermelho, sem verde ou azul.
 6. (255, 165, 0) - Laranja: Máximo de vermelho com verde moderado, sem azul.
 7. (0, 0, 255) - Azul: Máximo de azul, sem vermelho ou verde.
- Uso:
 - Cada peça é desenhada com uma cor aleatória desse array (`piece_color = random.choice(COLORS)`).
 - Quando fixada na grade, o índice da cor em `COLORS` (mais 1) é armazenado em `grid[y][x]` para referência.
 - A sombra da peça usa uma versão escurecida da cor (`color[0] // 2, color[1] // 2, color[2] // 2`).
 - Finalidade: Diferenciar visualmente as peças, mantendo consistência com o Tetris clássico, e criar contraste com o fundo.

3. Cores do Gradiente de Fundo

- Definidas em `draw_gradient_background([(0, 0, 0), (50, 50, 100)])`:
 - (0, 0, 0) - Preto: Ponto inicial do gradiente (topo da tela).
 - (50, 50, 100) - Azul Escuro: Ponto final do gradiente (base da tela), com baixa intensidade de vermelho e verde e azul moderado.
- Processo:
 - A função interpola linearmente entre essas cores para cada linha da tela, criando uma transição suave.
 - Exemplo: Para uma linha no meio da tela ($y = 400$), os valores RGB são calculados como uma média ponderada entre (0, 0, 0) e (50, 50, 100).
- Uso: Fundo de todas as telas (jogo, menu, recordes, etc.).
- Finalidade: Adiciona profundidade visual e evita um fundo monótono, destacando a grade e as peças.

4. Cores Derivadas

Algumas cores são geradas dinamicamente a partir das definidas:

- Texto 3D (`draw_3d_text`):
 - Camadas escuras: Cada canal RGB da cor base (ex.: `COLORS[0] = (0, 255, 255)`) é dividido por um fator crescente (`color[0] // (i + 1)`), criando um efeito de sombra.
 - Finalidade: Simula profundidade no título e textos importantes.
- Sombra da Peça:

- Cor escurecida: Divide os valores RGB da cor da peça por 2 (ex.: (0, 255, 255) vira (0, 127, 127)).
- Finalidade: Indica a posição final da peça com um tom mais sutil.
- Partículas (draw_particles):
 - Usa a mesma cor da peça fixada (piece_color), criando pequenos círculos em posições aleatórias.
 - Finalidade: Feedback visual ao fixar uma peça.

Ferramentas Usadas

- **Python:** Linguagem de programação base para lógica e estrutura do jogo.
- **Pygame:** Biblioteca para criação de jogos 2D, responsável por renderização gráfica, entrada de eventos e reprodução de som.
- **PressStart2P.ttf:** Fonte personalizada para textos, inspirada em estética retrô de jogos.
- **Tetris Theme.mp3:** Arquivo de áudio para música de fundo, carregado via Pygame Mixer.
- **Highscores.txt:** Arquivo de texto para armazenamento persistente de recordes (nome, pontuação e tempo).

Principais Técnicas de Computação Gráfica

Principais Técnicas:

1. Interpolação de Cores (Gradiente no Fundo)
 - Função: draw_gradient_background()
 - Descrição: Cria um fundo com transição suave entre duas cores (preto e azul escuro), calculando valores RGB intermediários para cada linha da tela.
 - Aplicação: Melhora a estética, destacando a grade do jogo com um efeito visual dinâmico.

Código:

```

48 # Função para desenhar gradiente no fundo
49 def draw_gradient_background(colors):
50     """
51     Desenha um gradiente de cores no fundo da tela.
52     Técnica de Computação Gráfica: Interpolação de cores.
53     """
54     for y in range(SCREEN_HEIGHT):
55         color = (
56             int(colors[0][0] + (colors[1][0] - colors[0][0]) * y / SCREEN_HEIGHT),
57             int(colors[0][1] + (colors[1][1] - colors[0][1]) * y / SCREEN_HEIGHT),
58             int(colors[0][2] + (colors[1][2] - colors[0][2]) * y / SCREEN_HEIGHT)
59         )
60         pygame.draw.line(screen, color, (0, y), (SCREEN_WIDTH, y))
61

```

2. Deslocamento e Escurecimento de Camadas (Texto 3D)

- Função: `draw_3d_text()`
- Descrição: Renderiza texto com efeito tridimensional ao desenhar camadas deslocadas com cores progressivamente mais escuras, simulando profundidade.
- Aplicação: Usado no título do menu e telas de dificuldade, conferindo um visual impactante.

Código:

```
62 # Função para desenhar texto 3D
63 def draw_3d_text(text, font, x, y, color, depth):
64     """
65     Desenha texto com efeito 3D.
66     Técnica de Computação Gráfica: Deslocamento e escurecimento de camadas.
67     """
68     for i in range(depth):
69         text_surface = font.render(text, True, (color[0] // (i + 1), color[1] // (i + 1), color[2] // (i + 1)))
70         screen.blit(text_surface, (x - i, y - i))
71     text_surface = font.render(text, True, color)
72     screen.blit(text_surface, (x, y))
```

3. Rasterização de Retângulos (Grade e Peças)

- Funções: `draw_grid()`, `draw_piece()`, `draw_shadow()`
- Descrição: Desenha retângulos para representar a grade, peças e suas sombras, utilizando coordenadas e cores específicas. A sombra usa transparência (cores divididas por 2) para indicar a posição final da peça.
- Aplicação: Base da visualização do jogo, permitindo a representação clara do estado da grade e das peças.

A rasterização de retângulos é realizada diretamente pela função `pygame.draw.rect()` da biblioteca Pygame. Essa função não especifica explicitamente um algoritmo clássico, pois ela encapsula a lógica de rasterização internamente. No entanto, podemos inferir o tipo de algoritmo comumente usados em bibliotecas como Pygame para desenhar retângulos preenchidos.

O algoritmo mais comum para rasterização de retângulos preenchidos, como os usados em `pygame.draw.rect()`, é o algoritmo de varredura por linha (Scanline Algorithm). Como ele funciona:

1. Definição dos Limites:

- O retângulo é definido por suas coordenadas (x, y) (canto superior esquerdo) e suas dimensões (width, height) (largura e altura).
- No caso do Tetris, essas coordenadas são calculadas como (x * BLOCK_SIZE, y * BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE).

2. Varredura Horizontal:

- O algoritmo "varre" cada linha de pixels dentro do retângulo, do y inicial até y + height.
- Para cada linha, ele preenche todos os pixels entre x e x + width com a cor especificada.

3. Preenchimento:

- Cada pixel na área do retângulo é atribuído à cor definida (ex.: `COLORS[grid[y][x] - 1]` para blocos fixos na grade).

- Isso é feito diretamente no buffer de pixels da tela, que é atualizado ao chamar `pygame.display.flip()`.

Código:

```

74 # Função para desenhar a grade
75 def draw_grid():
76     """
77     Desenha a grade do Tetris.
78     Técnica de Computação Gráfica: Rasterização de retângulos.
79     """
80     for y in range(GRID_HEIGHT):
81         for x in range(GRID_WIDTH):
82             if grid[y][x]:
83                 pygame.draw.rect(screen, COLORS[grid[y][x] - 1], (x * BLOCK_SIZE, y * BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE))
84             else:
85                 pygame.draw.rect(screen, GRAY, (x * BLOCK_SIZE, y * BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE), 1)
86
87 # Função para desenhar uma peça
88 def draw_piece(piece, x, y, color):
89     """
90     Desenha uma peça do Tetris.
91     Técnica de Computação Gráfica: Rasterização de retângulos.
92     """
93     for row in range(len(piece)):
94         for col in range(len(piece[row])):
95             if piece[row][col]:
96                 pygame.draw.rect(screen, color, ((x + col) * BLOCK_SIZE, (y + row) * BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE))
97
98 # Função para desenhar sombra da peça
99 def draw_shadow(piece, x, y, color):
100     """
101     Desenha a sombra da peça atual.
102     Técnica de Computação Gráfica: Rasterização de retângulos com transparência.
103     """
104     shadow_y = y
105     while not check_collision(piece, x, shadow_y + 1):
106         shadow_y += 1
107     for row in range(len(piece)):
108         for col in range(len(piece[row])):
109             if piece[row][col]:
110                 pygame.draw.rect(screen, (color[0] // 2, color[1] // 2, color[2] // 2), ((x + col) * BLOCK_SIZE, (shadow_y + row) * BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE))

```

4. Transformação Geométrica (Rotação de Peças)

- Função: `rotate_piece()`
- Descrição: Rotaciona uma matriz representando a peça usando transposição e inversão de linhas, garantindo que a rotação seja precisa e visualmente consistente.
- Aplicação: Essencial para a mecânica do Tetris, permitindo ao jogador ajustar as peças.

Código:

```

126 def rotate_piece(piece):
127     """
128     Rotaciona uma peça do Tetris.
129     Técnica de Computação Gráfica: Transformação geométrica (rotação de matriz).
130     """
131     return [list(row) for row in zip(*piece[::-1])]

```

5. Animação de Piscar (Remoção de Linhas)

- Função: `remove_complete_lines()`
- Descrição: Antes de remover linhas completas, exibe uma animação de piscar alternando entre branco e a cor original da linha por três ciclos.
- Aplicação: Feedback visual que destaca a eliminação de linhas, aumentando a satisfação do jogador.

Código:

```

133 # Função para remover linhas completas
134 def remove_completa_lines():
135     """
136     Remove linhas completas e exibe uma animação.
137     Técnica de Computação Gráfica: Animação de piscar.
138     """
139     lines_removed = 0
140     for row in range(GRID_HEIGHT):
141         if all(grid[row]):
142             for _ in range(3): # Animação de piscar
143                 for x in range(GRID_WIDTH):
144                     pygame.draw.rect(screen, WHITE, (x * BLOCK_SIZE, row * BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE))
145                 pygame.display.flip()
146                 pygame.time.wait(50)
147                 for x in range(GRID_WIDTH):
148                     pygame.draw.rect(screen, COLORS[grid[row][x] - 1], (x * BLOCK_SIZE, row * BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE))
149                 pygame.display.flip()
150                 pygame.time.wait(50)
151             del grid[row]
152             grid.insert(0, [0 for _ in range(GRID_WIDTH)])
153             lines_removed += 1
154     return lines_removed

```

6. Movimento com Trigonometria (Partículas)

- Função: draw_particles()
- Descrição: Gera partículas que se movem em direções aleatórias a partir de um ponto, usando seno e cosseno para calcular deslocamentos baseados em ângulos.
- Aplicação: Adiciona um efeito visual ao fixar peças na grade, tornando o jogo mais dinâmico.

Código:

```

156 # Função para desenhar partículas (efeito visual)
157 def draw_particles(x, y, color):
158     """
159     Desenha partículas para efeitos visuais.
160     Técnica de Computação Gráfica: Movimento com trigonometria.
161     """
162     for _ in range(20): # 20 partículas
163         angle = random.uniform(0, 2 * math.pi)
164         speed = random.uniform(1, 5)
165         dx = int(speed * math.cos(angle))
166         dy = int(speed * math.sin(angle))
167         pygame.draw.circle(screen, color, (x + dx, y + dy), 2)

```

7. Verificação de Limites e Sobreposição (Colisão)

- Função: check_collision()
- Descrição: Verifica se uma peça colide com bordas ou blocos existentes na grade, analisando coordenadas e ocupação de células.
- Aplicação: Garante a lógica do jogo, evitando sobreposições inválidas e determinando o fim da partida.

Código:

```

112 # Função para verificar colisão
113 def check_collision(piece, x, y):
114     """
115     Verifica se uma peça colide com a grade ou outras peças.
116     Técnica de Computação Gráfica: Verificação de limites e sobreposição.
117     """
118     for row in range(len(piece)):
119         for col in range(len(piece[row])):
120             if piece[row][col]:
121                 if y + row >= GRID_HEIGHT or x + col < 0 or x + col >= GRID_WIDTH or grid[y + row][x + col]:
122                     return True
123     return False

```

Telas do Jogo

Telas respectivamente: Menu Inicial, Dificuldade, Recordes, Pause, Colocar o Nome, e Em Jogo



Conclusões

O "TETRIS com Melhorias Visuais" combina com sucesso a jogabilidade clássica com técnicas modernas de Computação Gráfica, resultando em uma experiência visualmente rica e interativa. A implementação de gradientes, sombras e partículas eleva o apelo estético, enquanto animações e texto 3D reforçam a imersão. A estrutura modular do código facilita expansões futuras, como novos efeitos ou modos de jogo. Contudo, há espaço para melhorias, como otimização de desempenho em animações complexas e adição de mais opções de personalização (ex.: temas visuais). O projeto demonstra o potencial da Computação Gráfica em revitalizar jogos tradicionais, oferecendo uma base sólida para estudos ou desenvolvimentos adicionais.