

Resta Um (Versão C com Raylib) — Parte 3


Autor: Gabriel J Santos

Pontos Críticos e Confusos do Código

1 Coordenadas Cartesiana vs Tela

O código mantém **duas escalas de coordenadas**:

Tipo	Onde usado	Observações
Cartesiana	<code>parts[].posX</code> , <code>parts[].posY</code>	Lógica do tabuleiro, seleção, movimentação
Tela (pixels)	<code>DrawCircle()</code> , <code>DrawText()</code>	Renderização gráfica

 **Armadiilha comum:** Muitos iniciantes tentam usar coordenadas de tela diretamente na lógica do tabuleiro. Isso gera **bugs de seleção**, porque o mouse retorna pixels, não coordenadas lógicas.


Funções críticas:

- `scaled_to_screen()` → transforma coordenadas lógicas para pixels (com espaçamento).
- `scaled_to_screen_reverse()` → converte pixels para coordenadas lógicas (útil para movimentação futura).

2 Geração das Peças — Condição do `if`

```
if (!((i == 0) && (j == 0)) || ((abs(i) > 1) && (abs(j) > 1)))
```

- `(i == 0 && j == 0)` → remove o centro (posição inicial vazia).
- `(abs(i) > 1 && abs(j) > 1)` → remove cantos diagonais, deixando a **cruz central do tabuleiro**.

 **Sutil:** Esse `if` é o **coração do layout do jogo**. Alterar ou remover essa condição quebra a forma tradicional do “Resta Um”.

3 Estrutura de Seleção

```
struct selection {
    int Xcartesian;
    int Ycartesian;
    int X;
    int Y;
    bool state;
};
```

- **state** indica **se alguma peça foi selecionada**.
- **Sutil:** `get_selected()` atualiza `selected.state = false` para cada peça inativa, mas não reseta automaticamente quando nenhuma peça é clicada.
- Isso significa que, se o último loop não encontrar uma peça, **state** permanece **false**.

💡 **Dica:** sempre inicializar `selected.state = false` no começo da função para evitar erros invisíveis.

4 Detecção de Clique com Raio

```
if (get_distance(px, py, mouse.x, mouse.y) < R)
```

- Usa a **distância Euclidiana** para verificar se o clique está dentro do círculo da peça.
- **Sutil:** `R` define **tamanho de seleção**, não necessariamente o tamanho visual da peça.
- Ajustar `R` muito pequeno → clique pode “errar” a peça.
- Ajustar `R` muito grande → clique detecta peças próximas indevidamente.

5 Centralização de Texto

```
int text_width = MeasureText(title, fontSize);
DrawText(title, translate_cartesian_to_screen(0, screenSize.x) - (text_width/2),
10, fontSize, LIGHTGRAY);
```

- Medir o texto **antes de desenhar** é essencial para centralização.
- **Sutil:** `translate_cartesian_to_screen(0, screenSize.x)` retorna o **centro da tela**.
- Subtrair `text_width/2` garante que o texto fique **visualmente centralizado**.

💡 Se você remover `text_width/2`, o texto **não ficará alinhado com o tabuleiro**.

6 Laço de Desenho das Peças

```
for (int i = 0; i < 32; i++) {
    if (parts[i].state == true) {
        DrawCircle(scaled_to_screen(...), scaled_to_screen(...), R, LIGHTGRAY);
    }
}
```

- **Sutil:** mesmo que `parts[i]` esteja “invisível” (`state == false`), o índice continua existindo.
- Isso evita **crashes**, mas requer atenção se você for adicionar **movimento ou remoção dinâmica**.

7 Escala e Espaçamento

- **spacing** controla distância entre peças (px).
- **Sutil:** alterar **spacing** muda tanto:
 - posição de desenho (`scaled_to_screen()`)
 - cálculo de clique (`get_distance()`)
- Mudanças descoordenadas → peças desenhadas **fora do alcance do clique**.

8] Uso do **sqrt** no cálculo de distância

```
return sqrt((dx*dx) + (dy*dy));
```

- Retorna **um float**, mas a função está com retorno **int**.
- **Sutil:** perde precisão, mas não prejudica porque o raio R é grande o suficiente.
- Para maior precisão, a função poderia ser **float** `get_distance(...)`.

9] Loop de Atualização vs Desenho

- **Update** → captura o mouse, calcula seleção.
- **Draw** → desenha todo o tabuleiro **no estado atual**.
- **Sutil:** nunca desenhe fora de `BeginDrawing()/EndDrawing()`.
- Alterações no estado de peças devem ocorrer **antes do desenho**, caso contrário você verá **atrasos visuais**.

10] Dicas de Debug

1. Use `parts_print_console()` para verificar posições e estados.
2. Adicione prints dentro de `get_selected()` para ver quais peças estão sendo detectadas.
3. Visualize `Xcartesian` e `X` para entender como a **conversão de coordenadas** funciona.

Resumo dos Pontos Sussurros

Conceito	Sutileza / Erro Comum
Cartesiano vs Tela	Confusão leva a seleção incorreta
Condição de geração	Alterar quebra o formato da cruz
state em seleção	Não inicializar false → seleção "fantasma"
Raio de clique R	Muito grande ou pequeno → clique impreciso
Centralização do texto	Medir largura antes de desenhar é obrigatório
Laço de desenho	Ignorar state → peças invisíveis ainda existem

Conceito	Sutileza / Erro Comum
Escala/spacing	Afeta clique e desenho simultaneamente
<code>sqrt</code> retornando int	Perde precisão discreta
Ordem Update/Draw	Alterações no estado devem ocorrer antes do desenho

💡 **Resumo final da Parte 3:** Esses pontos são **onde o código esconde armadilhas e detalhes sutis**. Compreender essas relações é **crucial** antes de adicionar funcionalidades, como movimentação de peças ou contagem de peças restantes.