

UNIVERSIDADE FEDERAL DE MINAS GERAIS



Daiana David Rodrigues

DESENVOLVIMENTO DE JOGO ELETRÔNICO: SPACE INVADERS

UMA IMPLEMENTAÇÃO EM C COM A BIBLIOTECA ALLEGRO

BELO HORIZONTE

2025

Daiana David RODRIGUES

DESENVOLVIMENTO DE JOGO ELETRÔNICO: SPACE INVADERS

UMA IMPLEMENTAÇÃO EM C COM A BIBLIOTECA ALLEGRO

Trabalho final apresentado à disciplina de Programação e Desenvolvimento de Softwares do Departamento de Ciências da Computação da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção de nota.

Orientadores: Prof. Adriano C. M. Pereira

BELO HORIZONTE

2025

RESUMO

Este trabalho detalha o processo de desenvolvimento de uma recriação do clássico jogo de arcade *Space Invaders*, utilizando a linguagem de programação C e a biblioteca gráfica Allegro 5. O projeto abrange desde a configuração do ambiente de desenvolvimento até a implementação de mecânicas de jogo mais elaboradas, como a gestão de múltiplas telas de jogo (menu, gameplay, game over), renderização de sprites a partir de um *spritesheet*, movimentação síncrona de uma horda de inimigos, detecção de colisões e armazenamento de dados através do salvamento de recordes em um documento de texto. O resultado é um jogo funcional que não apenas cumpre os requisitos técnicos propostos, mas também oferece uma experiência para o usuário alternativa e única (pois foi uma criação totalmente personalizada), com recursos visuais e sonoros que remetem ao jogo original.

Palavras-chave: Desenvolvimento de Jogos, Linguagem C, Allegro 5, Space Invaders, Spritesheet.

ABSTRACT

This work details the development process of a recreation of the classic arcade game Space Invaders, using the C programming language and the Allegro 5 graphics library. The project covers everything from setting up the development environment to implementing more elaborate game mechanics, such as managing multiple game screens (menu, gameplay, game over), rendering sprites from a spritesheet, synchronized movement of an enemy horde, collision detection, and data storage through saving records in a text document. The result is a fully functional game that not only meets the proposed technical requirements but also offers a unique and alternative user experience (as it was a completely personalized creation), with visual and sound features reminiscent of the original game.

Keywords: Game Development, C Language, Allegro 5, Space Invaders, Spritesheet, State Machine.

SUMÁRIO

1 INTRODUÇÃO	10
1.1 Objetivos	10
2 FUNDAMENTAÇÃO TEÓRICA	10
2.1 Bibliotecas da Linguagem C Utilizadas	10
2.1.1 Biblioteca stdio.h	10
2.1.2 Biblioteca stdlib.h	10
2.1.3 Biblioteca time.h	11
2.1.4 Biblioteca stdbool.h	11
2.2 Bibliotecas do Allegro 5	11
3 ARQUITETURA DO SISTEMA	11
3.1 Constantes Globais	11
3.1.1 Configurações de Performance	11
3.1.2 Configurações de Gameplay	11
3.1.3 Dimensões dos Sprites	12
3.2 Estruturas de Dados	12
3.2.1 Enumeração Estado Jogo	12
3.2.2 Estrutura Nave	12
3.2.3 Estrutura Alien	12
3.2.4 Estrutura Tiro	13
3.2.5 Estrutura Pontuacao	13
4 IMPLEMENTAÇÃO	13
4.1 Funções de Inicialização	13
4.1.1 Inicialização da Nave	13
4.1.2 Inicialização da Horda de Aliens	13
4.1.3 Inicialização do Sistema de Pontuação	13
4.1.4 Inicialização dos Objetos de Jogo	14
4.2 Funções de Atualização	14
4.2.1 Atualização da Nave	14
4.2.2 Atualização da Horda	14
4.2.3 Atualização do Tiro	14
4.3 Sistema de Detecção de Colisões	15
4.3.1 Teoria das Bounding Boxes	15
4.3.2 Implementação: Tiro vs. Alien	15
4.3.3 Implementação: Aliens vs. Nave	16
4.3.4 Implementação: Aliens vs. Solo	17
4.3.5 Implementação: Aliens Executados	17
4.4 Sistema de Persistência de Dados	17
4.4.1 Atualização de Recordes	17
4.5 Loop Principal e Máquina de Estados	18

4.5.1 Estrutura do Loop Principal	18
4.5.2 Lógica do Estado MENU	18
4.5.3 Lógica do Estado JOGANDO	19
4.5.4 Lógica do Estado FIM_DE_JOGO	19
4.5.5 Lógica de Desenho	19
4.5.6 Finalização e Limpeza de Memória	19
5 SISTEMA DE RENDERIZAÇÃO	20
5.1 Funções de Desenho	20
5.1.1 Desenho do Cenário	20
5.1.2 Desenho da Nave	20
5.1.3 Desenho da Horda de Aliens	21
5.1.4 Desenho do Tiro	22
5.1.5 Desenho da Pontuação	22
5.2 Paleta de Cores	23
5.3 Desenho das Telas	23
5.3.1 Tela de Menu	23
5.3.2 Tela de Fim de Jogo	24
6 REFERÊNCIA DA BIBLIOTECA ALLEGRO 5	24
6.1 Funções de Inicialização	24
6.2 Gerenciamento de Display	25
6.3 Manipulação de Bitmaps	25
6.4 Sistema de Eventos	25
6.5 Códigos de Eventos Principais	26
7. CONCLUSÃO	26
7.1 Contribuições Técnicas	26
7.2 Trabalhos Futuros	26
8. REFERÊNCIAS	26

1 INTRODUÇÃO

O Space Invaders é um dos jogos arcade mais icônicos da história dos videogames, criado por Tomohiro Nishikado em 1978. Este projeto consiste na recriação deste clássico utilizando a linguagem de programação C em conjunto com a biblioteca gráfica Allegro 5, proporcionando uma experiência de aprendizado completa.

1.1 Objetivos

- Implementar um jogo funcional Space Invaders em linguagem C
- Demonstrar o uso prático da biblioteca Allegro 5
- Aplicar conceitos fundamentais de programação estruturada
- Controle preciso dos movimentos da nave
- O grupo de naves deve ter ao menos 4 linhas e 5 colunas
- Deve haver ao menos 30 pixels de espaço entre as naves
- As naves devem se mover juntas, conforme o comportamento do jogo original
- O jogo deve permitir apenas um tiro por vez
- O tiro deve eliminar a nave ao colidir com ela
- Se qualquer nave tocar o solo, o jogo deve terminar
- Se qualquer nave colidir (ou estiver muito próxima) do canhão, o jogo deve terminar
- O tiro deve ser vertical, sem mudança de direção ou angulação
- O cenário deve exibir a pontuação do jogador
- Exibição e armazenamento do recorde
- Documentação

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Bibliotecas da Linguagem C Utilizadas

2.1.1 Biblioteca stdio.h

Biblioteca padrão de entrada e saída, fornecendo funcionalidades para interação com arquivos e terminal.

Funções principais:

Função	Escopo	Descrição
printf	(formato, argumentos)	Impressão formatada na tela
fopen	(nome_arquivo, modo)	Abertura de arquivos
fclose	(ponteiro_arquivo)	Fechamento de arquivos

fscanf	(arquivo, formato, variável)	Leitura formatada de arquivos
fprintf	(arquivo, formato, argumentos)	Escrita formatada em arquivos

2.1.2 Biblioteca stdlib.h

Biblioteca para gerenciamento de memória e funções utilitárias.

Funções utilizadas:

- `rand()`: Geração de números pseudo-aleatórios
- `srand(semente)`: Inicialização do gerador de números aleatórios

2.1.3 Biblioteca time.h

Biblioteca para manipulação de tempo e data.

Função utilizada:

- `time(NULL)`: Obtenção do timestamp atual para semente aleatória

2.1.4 Bibliotecastdbool.h

Adiciona o tipo de dado `bool` à linguagem C, permitindo uso de `true` e `false`.

2.2 Bibliotecas do Allegro 5

- `allegro5/allegro.h`: Funcionalidades básicas do Allegro
- `allegro5/allegro_audio.h`: Sistema de áudio
- `allegro5/allegro_acodec.h`: Codecs de áudio para diferentes formatos

3 ARQUITETURA DO SISTEMA

3.1 Constantes Globais

A definição de constantes globais segue boas práticas de programação, facilitando a manutenção e ajustes posteriores.

3.1.1 Configurações de Performance

- `const float FPS = 60`

Define que o jogo deve tentar rodar a 60 quadros por segundo (frames per second). O timer do Allegro usa esse valor para saber com que frequência ele precisa disparar o evento de atualização.

- **const int SCREEN_W = 960 e const int SCREEN_H = 540**

Definem a largura (width) e a altura (height) da janela do jogo, em pixels. Essas constantes são usadas na criação da tela e também para controlar os limites visuais dela.

3.1.2 Configurações de Gameplay

- **const int GRASS_H = 60:**

No jogo base mostrado nos vídeos do professor Pedro Olmo, essa constante era usada para desenhar a grama com a função `draw_rectangle` da biblioteca Allegro. No entanto, como foi usada uma imagem no lugar, essa constante passou a funcionar como limite da zona de perigo, ou seja, representa a altura da imagem da grama na tela. A função de colisão com o solo usa esse valor como referência.

3.1.3 Dimensões dos Sprites

- **const int NAVE_W = 73 e const int NAVE_H = 100**

Definem a largura e a altura da sprite da nave. Isso é usado tanto para posicionamento correto na tela quanto para limitar o movimento até as bordas e detectar colisões.

- **const int ALIEN_W = 90 e const int ALIEN_H = 79**

São as dimensões de recorte de um único alien dentro da imagem `catalogo_inimigos.png`. Essas medidas são importantes para extrair corretamente cada sprite da imagem.

- **const int ESP_X = 19 e const int ESP_Y = 15**

Foram medidas manualmente e representam o espaçamento entre os aliens no arquivo de imagem `catalogo_inimigos.png`. Elas são usadas na função `draw_horda` para calcular certinho o deslocamento entre um sprite e outro ao desenhar na tela com `al_draw_scaled_bitmap`.

- **const int ALIEN_DISPLAY_W = 50 e const int ALIEN_DISPLAY_H = 44**

São as dimensões finais de exibição dos aliens. Isso define o tamanho que cada alien vai ter na tela depois de ser redimensionado com a função `al_draw_scaled_bitmap`.

- **const int TIRO_W = 10 e const int TIRO_H = 17**

Definem a largura e a altura da imagem do tiro. Essas dimensões são essenciais para detectar colisões com os aliens e também para centralizar corretamente o tiro no momento em que ele é disparado.

3.2 Estruturas de Dados

3.2.1 Enumeração Estado Jogo

```
39  typedef enum {
40      MENU,
41      JOGANDO,
42      FIM_DE_JOGO
43  } EstadoJogo;
```

Esta enumeração define os possíveis estados do jogo, facilitando o controle de fluxo entre diferentes telas e lógicas de gameplay.

3.2.2 Estrutura Nave

```
46  typedef struct Nave {
47      float x;
48      float y;
49      float velocidade;
50      int dir, esq;
51      ALLEGRO_BITMAP *sprite_da_nave;
52  } Nave;
```

Descrição dos campos:

- **x, y**: Coordenadas cartesianas de posição
- **velocidade**: Taxa de deslocamento em pixels por quadro
- **dir, esq**: Variáveis de controle para movimento contínuo
- **sprite_da_nave**: Referência para a imagem carregada em memória

3.2.3 Estrutura Alien

```
89  typedef struct Alien {
90      float x, y;
91      bool vivo;
92      int sprite_coluna;
93      int sprite_linha;
94  } Alien;
```

Descrição dos campos:

- **x, y**: Coordenadas de posição
- **vivo**: Flag booleana indicando se o alien está ativo
- **sprite_coluna, sprite_linha**: Coordenadas para recorte do spritesheet

3.2.4 Estrutura Tiro

```

188  typedef struct Tiro {
189      float x, y;
190      float vel;
191      bool ativo;
192      bool explodindo;
193      int tempo_explosao;
194      ALLEGRO_BITMAP *explosao_sprite;
195      ALLEGRO_BITMAP *sprite_bala;
196  } Tiro;

```

Descrição dos campos:

- **x, y**: Posição atual do tiro na tela
- **vel**: Velocidade com que o tiro se move
- **ativo**: Flag que indica se o tiro está ativo (em movimento) ou não
- **explodindo**: Flag que indica se o tiro está no meio da animação de explosão
- **tempo_explosao**: Contador que controla por quanto tempo a animação de explosão vai durar
- **explosao_sprite**: Ponteiro para a imagem da explosão
- **sprite_bala**: Ponteiro para a imagem do projétil

3.2.5 Estrutura Pontuacao

```

280  typedef struct {
281      int pontuacao_atual;
282      int recorde;
283  } Pontuacao;
284

```

Descrição dos campos:

- **pontuacao_atual**: Armazena a pontuação acumulada durante a partida atual
- **recorde**: Guarda a maior pontuação registrada até o momento, servindo como um "high score"

4 IMPLEMENTAÇÃO

4.1 Funções de Inicialização

4.1.1 Inicialização da Nave

```

54 void initNave(Nave *nave) {
55     nave->x = SCREEN_W/2 - NAVE_W/2;
56     nave->velocidade = 2.5;
57     nave->dir = 0;
58     nave->esq = 0;
59     nave->y = SCREEN_H - GRASS_H - NAVE_H;
60
61     nave->sprite_da_nave = al_load_bitmap("navesprite.png");
62     if(!nave->sprite_da_nave) {
63         fprintf(stderr, "Falha ao carregar o sprite da nave!\n");
64         exit(1);
65     }
66 }
```

A função update_nave verifica se as flags dir (direita) ou esq (esquerda) estão ativadas. Se estiver indo para direita e ainda não tiver encostado no limite da tela (SCREEN_W - 90), a nave anda somando a velocidade à posição x. O valor 90 serve como margem de segurança para o lado direito, geralmente usado quando a largura da sprite não é exata ou por questão visual. Mesmo que NAVE_W não seja 90, esse número um pouco maior evita que a imagem "grude" na borda. O mesmo vale para esquerda: se ainda tiver espaço (x - velocidade >= 0), a nave anda para esquerda.

4.1.2 Inicialização da Horda de Aliens

```

96 void init_horda(Alien inimigos[4][5]) {
97     int i, j;
98     int esp_x = ALIEN_DISPLAY_W + 30;
99     int esp_y = ALIEN_DISPLAY_H + 20;
100    int margens_x = 50;
101    int margens_y = 40;
102
103    for(i = 0; i < 4; i++) {
104        for(j = 0; j < 5; j++) {
105            Alien* alien_atual = &inimigos[i][j];
106            alien_atual->vivo = true;
107            alien_atual->x = margens_x + j * esp_x;
108            alien_atual->y = margens_y + i * esp_y;
109            alien_atual->sprite_linha = rand() % 2;
110            alien_atual->sprite_coluna = rand() % 3;
111        }
112    }
113 }
```

Esta função configura a posição inicial e as propriedades de cada alien na matriz 4x5 que representa a formação dos inimigos.

4.1.3 Inicialização do Sistema de Pontuação

```

285 void initPontuacao(Pontuacao *pont) {
286     pont->pontuacao_atual = 0;
287     FILE *arquivo = fopen("recorde.txt", "r");
288     if(arquivo) {
289         fscanf(arquivo, "%d", &pont->recorde);
290         fclose(arquivo);
291     } else {
292         pont->recorde = 0;
293     }
294 }
```

Algoritmo de persistência:

1. Inicializa pontuação atual como zero

2. Tenta abrir arquivo de recordes em modo leitura
3. Se bem-sucedido, lê valor do recorde
4. Caso contrário, define recorde como zero

4.1.4 Inicialização dos Objetos de Jogo

Depois de todos os recursos estarem prontos, nós finalmente criamos e preparamos os nossos objetos de jogo.

- **Nave nave;**, **Alien inimigos[4][5];**, etc.: Declaramos as nossas variáveis de struct.
- **initNave(&nave);**, **init_horda(inimigos);**, etc.: Chamamos as nossas funções de inicialização, passando o endereço de memória (&) das structs para que os seus valores iniciais sejam configurados.

4.2 Funções de Atualização

4.2.1 Atualização da Nave

```

77 void update_nave(Nave *nave) {
78     if(nave->dir && nave->x + nave->velocidade <= SCREEN_W - 90) {
79         nave->x += nave->velocidade;
80     }
81
82     if(nave->esq && nave->x - nave->velocidade >= 0) {
83         nave->x -= nave->velocidade;
84     }
85 }
```

A função `update_nave` verifica se as flags `dir` (direita) ou `esq` (esquerda) estão ativadas. Se estiver indo para direita e ainda não tiver encostado no limite da tela (`SCREEN_W - 90`), a nave anda somando a velocidade à posição x. O valor 90 serve como margem de segurança para o lado direito, geralmente usado quando a largura da sprite não é exata ou por questão visual. Mesmo que `NAVE_W` seja 73, esse número um pouco maior evita que a imagem "grude" na borda. O mesmo vale para esquerda: se ainda tiver espaço (`x - velocidade >= 0`), a nave anda para esquerda.

4.2.2 Atualização da Horda

```
134 void update_horda(Alien inimigos[4][5], float *vel_horda_x) {
135     int i, j;
136     bool bateu_borda = false;
137
138     for(i = 0; i < 4; i++) {
139         for(j = 0; j < 5; j++) {
140             if(inimigos[i][j].vivo) {
141                 inimigos[i][j].x += (*vel_horda_x);
142             }
143         }
144     }
145
146     for(i = 0; i < 4; i++) {
147         for(j = 0; j < 5; j++) {
148             if(inimigos[i][j].vivo) {
149                 if((inimigos[i][j].x + ALIEN_DISPLAY_W > SCREEN_W) || (inimigos[i][j].x < 0)) {
150                     bateu_borda = true;
151                     break;
152                 }
153             }
154         }
155         if(bateu_borda) break;
156     }
157
158     if(bateu_borda) {
159         *vel_horda_x *= -1;
160         for(i = 0; i < 4; i++) {
161             for(j = 0; j < 5; j++) {
162                 if(inimigos[i][j].vivo) {
163                     inimigos[i][j].y += ALIEN_DISPLAY_H/2;
164                 }
165             }
166         }
167     }
168 }
```

Lógica da movimentação da horda:

1. Primeiro, todos os aliens vivos se movem na horizontal, somando a velocidade vel_horda_x
2. Depois, o código verifica se algum alien chegou na borda da tela (direita ou esquerda). Se sim, flag bateu_borda é ativada
3. Se a borda foi atingida, a direção é invertida (multiplica por -1) e todos os aliens vivos descem meio "degrau", ou seja, metade da altura deles
4. O break dentro do for é usado para parar de checar logo que o primeiro alien bater na borda, economizando processamento

4.2.3 Atualização do Tiro

```
207 void update_tiro(Tiro *tiro) {
208     if(tiro->ativo) {
209         tiro->y += tiro->vel;
210         if(tiro->y < 0) {
211             tiro->ativo = false;
212         }
213     }
214
215     if(tiro->explodindo) {
216         tiro->tempo_explosao--;
217         if(tiro->tempo_explosao <= 0) {
218             tiro->explodindo = false;
219         }
220     }
221 }
```

Lógica do projétil:

- Se o tiro está ativo, ele se move verticalmente, somando sua velocidade à coordenada y
- Quando $y < 0$, significa que o tiro já saiu da tela. Nesse caso, ele é desativado (ativo = false) para que possa ser reutilizado depois
- A função não trata colisão com inimigos, isso acontece em outra parte do código, geralmente no update_geral ou algo semelhante

4.3 Sistema de Detecção de Colisões

4.3.1 Teoria das Bounding Boxes

A detecção de colisão utiliza o algoritmo de Axis-Aligned Bounding Box (AABB), onde cada sprite é representado por um retângulo delimitador. A colisão ocorre quando há sobreposição em ambos os eixos cartesianos.

4.3.2 Implementação: Tiro vs. Alien

```

235 bool verifica_colisao_tiro_alien(Tiro *tiro, Alien inimigos[4][5]) {
236     if(!tiro->ativo) return false;
237     int i, j;
238
239     for(i = 0; i < 4; i++) {
240         for(j = 0; j < 5; j++) {
241             if(inimigos[i][j].vivo) {
242                 if(tiro->x < inimigos[i][j].x + ALIEN_DISPLAY_W &&
243                     tiro->x + TIRO_W > inimigos[i][j].x &&
244                     tiro->y < inimigos[i][j].y + ALIEN_DISPLAY_H &&
245                     tiro->y + TIRO_H > inimigos[i][j].y) {
246
247                     inimigos[i][j].vivo = false;
248                     tiro->ativo = false;
249                     tiro->explodindo = true;
250                     tiro->tempo_explosao = 30;
251                     return true;
252                 }
253             }
254         }
255     }
256     return false;
257 }
258

```

Fluxo de execução:

1. Verifica se o tiro está ativo
2. Itera sobre todos os aliens vivos
3. Aplica algoritmo AABB para cada alien
4. Em caso de colisão, executa ações consequentes

4.3.3 Implementação: Aliens vs. Nave

```
261 bool verifica_colisao_alien_nave(Alien inimigos[4][5], Nave *nave) {
262     int i, j;
263     for(i = 0; i < 4; i++) {
264         for(j = 0; j < 5; j++) {
265             if(inimigos[i][j].vivo) {
266                 if(inimigos[i][j].x < nave->x + NAVE_W &&
267                     inimigos[i][j].x + ALIEN_DISPLAY_W > nave->x &&
268                     inimigos[i][j].y < nave->y + NAVE_H &&
269                     inimigos[i][j].y + ALIEN_DISPLAY_H > nave->y) {
270                     return true;
271                 }
272             }
273         }
274     }
275     return false;
276 }
```

Fluxo de execução:

1. Verifica se a nave está ativa
2. Itera sobre todos os aliens vivos
3. Aplica algoritmo AABB para cada alien
4. Em caso de colisão, executa ações consequentes (fim de jogo)

4.3.4 Implementação: Aliens vs. Solo

```
172 bool verifica_colisao_horda_solo(Alien inimigos[4][5]) {
173     int i, j;
174     int topo_do_chao = SCREEN_H - 130;
175
176     for(i = 0; i < 4; i++) {
177         for(j = 0; j < 5; j++) {
178             if(inimigos[i][j].vivo && (inimigos[i][j].y + ALIEN_DISPLAY_H > topo_do_chao)) {
179                 return true;
180             }
181         }
182     }
183     return false;
184 }
```

Fluxo de execução:

1. Verifica se o sistema de solo está ativo
2. Itera sobre todos os aliens vivos
3. Aplica algoritmo AABB para cada alien (verificando se $y \geq SCREEN_H - GRASS_H$)
4. Em caso de colisão, executa ações consequentes (condição de derrota)

4.3.5 Implementação: Aliens Executados

```
353     bool todos_aliens_mortos(Alien inimigos[4][5]) {
354         int i, j;
355         for(i = 0; i < 4; i++) {
356             for(j = 0; j < 5; j++) {
357                 if(inimigos[i][j].vivo) {
358                     return false;
359                 }
360             }
361         }
362         return true;
363     }
```

Fluxo de execução:

1. Verifica se o sistema de vitória está ativo
2. Itera sobre todos os aliens da matriz
3. Aplica verificação de estado para cada alien (vivo = false)
4. Em caso de todos eliminados, executa ações consequentes (vitória do jogador)

4.4 Sistema de Persistência de Dados

4.4.1 Atualização de Recordes

```
296     void update_recorde(Pontuacao *pont) {
297         if(pont->pontuacao_atual > pont->recorde) {
298             pont->recorde = pont->pontuacao_atual;
299             FILE *arquivo = fopen("recorde.txt", "w");
300             if(arquivo) {
301                 fprintf(arquivo, "%d", pont->recorde);
302                 fclose(arquivo);
303             }
304         }
305     }
```

Lógica de verificação e salvamento:

1. A função compara a pontuacao_atual com o recorde atual
2. Se a pontuação da partida for maior que o recorde salvo, significa que o jogador quebrou o recorde, então a variável recorde é atualizada
3. Depois disso, o valor é salvo no arquivo recorde.txt usando o modo de escrita "w", que sobrescreve o conteúdo anterior com o novo recorde
4. O fopen é protegido com if(arquivo) para garantir que o arquivo abriu corretamente antes de escrever
5. Esta função normalmente é chamada no final da partida, ou no momento em que se detecta que a pontuação aumentou

4.5 Loop Principal e Máquina de Estados

4.5.1 Estrutura do Loop Principal

O `while(rodando)` é o coração do jogo, que fica repetindo-se até que o jogador decida sair.

- **bool redraw = true:** Esta é a flag de controle para o desenho. Ela é usada para otimizar o jogo. A lógica é: só precisamos redesenhar a tela se algo mudou. A redraw é definida como true sempre que o timer dispara.
- **ALLEGRO_EVENT ev:** Declara uma estrutura de evento vazia, como uma "carta" em branco.
- **al_wait_for_event(event_queue, &ev):** Esta função é o que faz o jogo "pausar" e esperar por uma ação. Ele para aqui até que uma nova "carta" (um evento, como um toque de tecla ou um pulso do timer) chegue na nossa "caixa de correio" (event_queue). Quando o evento chega, as suas informações são colocadas na variável ev.
- **switch(estado_atual):** Este é o "seletor de canais". Ele lê o valor atual da estado_atual e executa apenas o código que está dentro do case correspondente.

4.5.2 Lógica do Estado MENU

Quando estado_atual é MENU, o jogo só se preocupa com uma coisa: se o jogador apertou ENTER. Se o evento for ALLEGRO_KEY_DOWN e a tecla for ALLEGRO_KEY_ENTER, ele executa o bloco de reinicialização do jogo e muda o estado para JOGANDO.

4.5.3 Lógica do Estado JOGANDO

Este é o estado principal. Aqui, a cadeia if/else if verifica o tipo de evento:

- Se for ALLEGRO_EVENT_TIMER, ele chama todas as funções update... e verifica_colisao.... Se uma colisão de derrota acontecer, ele salva o recorde e muda o estado_atual para FIM_DE_JOGO.
- Se for ALLEGRO_EVENT_KEY_DOWN ou ALLEGRO_EVENT_KEY_UP, ele executa a lógica de movimento da nave e de tiro.

4.5.4 Lógica do Estado FIM_DE_JOGO

Aqui, o jogo apenas espera pelas teclas R (para reiniciar, o que envolve chamar o bloco de reinicialização e mudar o estado) ou ESC (para sair, o que define rodando = false).

4.5.5 Lógica de Desenho

Esta parte do código está no final do while, dentro de um if.

if(redraw && al_is_event_queue_empty(event_queue)): Esta condição é uma otimização de performance. Ela significa: "Só desenhe a tela se (&&) duas coisas forem verdadeiras:

- 1) redraw for true (o timer disparou e algo pode ter mudado de posição)
- 2) a fila de eventos estiver vazia (al_is_event_queue_empty)". Esta segunda parte é importante para garantir que todos os eventos de input (como vários toques de tecla rápidos) sejam processados antes de desenhar a tela, evitando atrasos visuais.

redraw = false: Logo que entramos no bloco de desenho, definimos redraw como false. Isto impede que o jogo redesenhe a mesma coisa várias vezes desnecessariamente até o próximo pulso do timer.

if/else if (estado_atual == ...): Um if/else if simples verifica o estado atual e chama apenas as funções de desenho corretas para aquela tela.

al_flip_display(): Esta é a última etapa. Depois de todas as funções de desenho terem "pintado" na memória, esta função pega o resultado final e o exibe no monitor do jogador.

4.5.6 Finalização e Limpeza de Memória

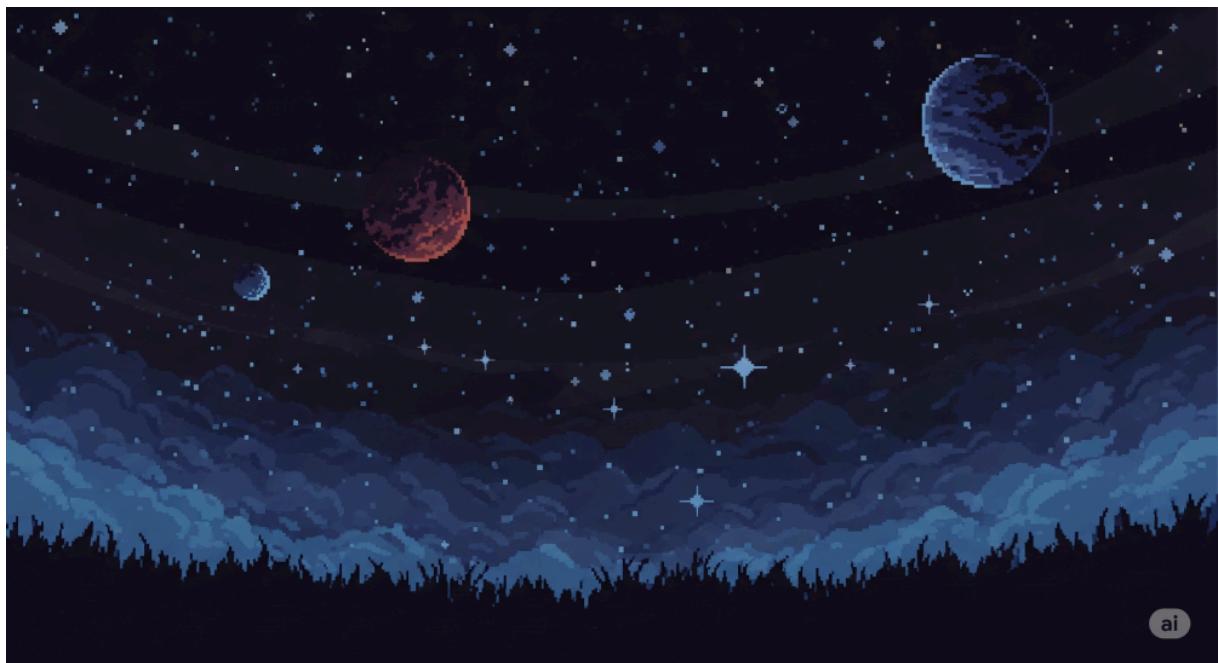
Quando o while(rodando) termina (porque rodando se tornou false), o programa executa o código final.

al_destroy_...(): Para cada recurso que foi criado com al_create_... ou carregado com al_load_..., existe uma função al_destroy_... correspondente. É crucial chamar todas elas no final para libertar a memória que o jogo usou e devolvê-la ao sistema operativo, garantindo que o programa feche de forma limpa e sem "memory leaks".

5 SISTEMA DE RENDERIZAÇÃO

5.1 Funções de Desenho

5.1.1 Desenho do Cenário



Fonte: Imagem gerada por Gemini.

```
69 void draw_scenario(ALLEGRO_BITMAP *Fundo) {  
70     al_draw_bitmap(fundo, 0, 0, 0);  
71 }
```

Esta função desenha o bitmap do fundo na posição (0,0) da tela, cobrindo toda a área do display. O último parâmetro (0) indica que não há flags especiais de desenho aplicadas

5.1.2 Desenho da Nave



Fonte: Imagem gerada por Gemini.

```

73 void draw_nave(Nave nave) {
74     al_draw_bitmap(nave.sprite_da_nave, nave.x, nave.y, 0);
75 }
76

```

Desenha a sprite da nave nas coordenadas x e y armazenadas na estrutura. A função `al_draw_bitmap` renderiza a imagem sem transformações adicionais.

5.1.3 Desenho da Horda de Aliens



Fonte: Imagem gerada por Gemini.

```

115 void draw_horda(Alien inimigos[4][5], ALLEGRO_BITMAP *spritesheet) {
116     int i, j;
117     for(i = 0; i < 4; i++) {
118         for(j = 0; j < 5; j++) {
119             if(inimigos[i][j].vivo) {
120                 float sprite_x = inimigos[i][j].sprite_coluna * (ALIEN_W + ESP_X);
121                 float sprite_y = inimigos[i][j].sprite_linha * (ALIEN_H + ESP_Y);
122
123                 al_draw_scaled_bitmap(spritesheet,
124                                         sprite_x, sprite_y,
125                                         ALIEN_W, ALIEN_H,
126                                         inimigos[i][j].x, inimigos[i][j].y,
127                                         ALIEN_DISPLAY_W, ALIEN_DISPLAY_H,
128                                         0);
129             }
130         }
131     }
132 }

```

Esta função percorre a matriz de aliens e, para cada alien vivo, calcula a posição do sprite no spritesheet. As variáveis `sprite_x` e `sprite_y` determinam o recorte correto da imagem baseado nas coordenadas `sprite_coluna` e `sprite_linha` do alien.

A função `al_draw_scaled_bitmap` possui os seguintes parâmetros:

- **spritesheet**: Bitmap fonte
- **sprite_x, sprite_y**: Coordenadas de origem no spritesheet
- **ALIEN_W, ALIEN_H**: Dimensões do recorte no spritesheet

- `inimigos[i][j].x, inimigos[i][j].y`: Posição de destino na tela
- `ALIEN_DISPLAY_W, ALIEN_DISPLAY_H`: Dimensões finais na tela
- `0`: Flags de desenho

5.1.4 Desenho do Tiro



Fonte: Imagem gerada por Gemini.

```

223 void draw_tiro(Tiro tiro) {
224     if(tiro.ativo && tiro.sprite_bala) {
225         al_draw_bitmap(tiro.sprite_bala, tiro.x, tiro.y, 0);
226     }
227
228     if(tiro.explodindo && tiro.explosao_sprite) {
229         al_draw_bitmap(tiro.explosao_sprite, tiro.x - 17, tiro.y - 17, 0);
230     }
231 }
```

Esta função desenha o tiro quando ele está ativo e a explosão quando o tiro está explodindo. O valor 17 nas coordenadas da explosão serve para centralizar a animação de explosão em relação à posição original do tiro, já que a sprite de explosão é maior que a sprite do projétil.

5.1.5 Desenho da Pontuação

```
307 void draw_pontuacao(Pontuacao pont, ALLEGRO_FONT *font) {  
308     if(font) {  
309         // Sombra para RECORD  
310         al_draw_textf(font, al_map_rgb(0, 0, 0), SCREEN_W - 9, 9, ALLEGRO_ALIGN_RIGHT, "RECORD: %d", pont.recorde);  
311         al_draw_textf(font, al_map_rgb(0, 0, 0), SCREEN_W - 11, 9, ALLEGRO_ALIGN_RIGHT, "RECORD: %d", pont.recorde);  
312         al_draw_textf(font, al_map_rgb(0, 0, 0), SCREEN_W - 9, 11, ALLEGRO_ALIGN_RIGHT, "RECORD: %d", pont.recorde);  
313         al_draw_textf(font, al_map_rgb(0, 0, 0), SCREEN_W - 11, 11, ALLEGRO_ALIGN_RIGHT, "RECORD: %d", pont.recorde);  
314  
315         // Texto principal  
316         al_draw_textf(font, al_map_rgb(41, 74, 183), 10, 10, 0, "SCORE: %d", pont.pontuacao_atual);  
317         al_draw_textf(font, al_map_rgb(41, 74, 183), SCREEN_W - 10, 10, ALLEGRO_ALIGN_RIGHT, "RECORD: %d", pont.recorde);  
318     }  
319 }
```

Esta função desenha a pontuação atual e o recorde na tela. Primeiro desenha múltiplas versões do texto do recorde em preto (sombra) com pequenos deslocamentos para criar um efeito de profundidade. Depois desenha os textos principais em azul.

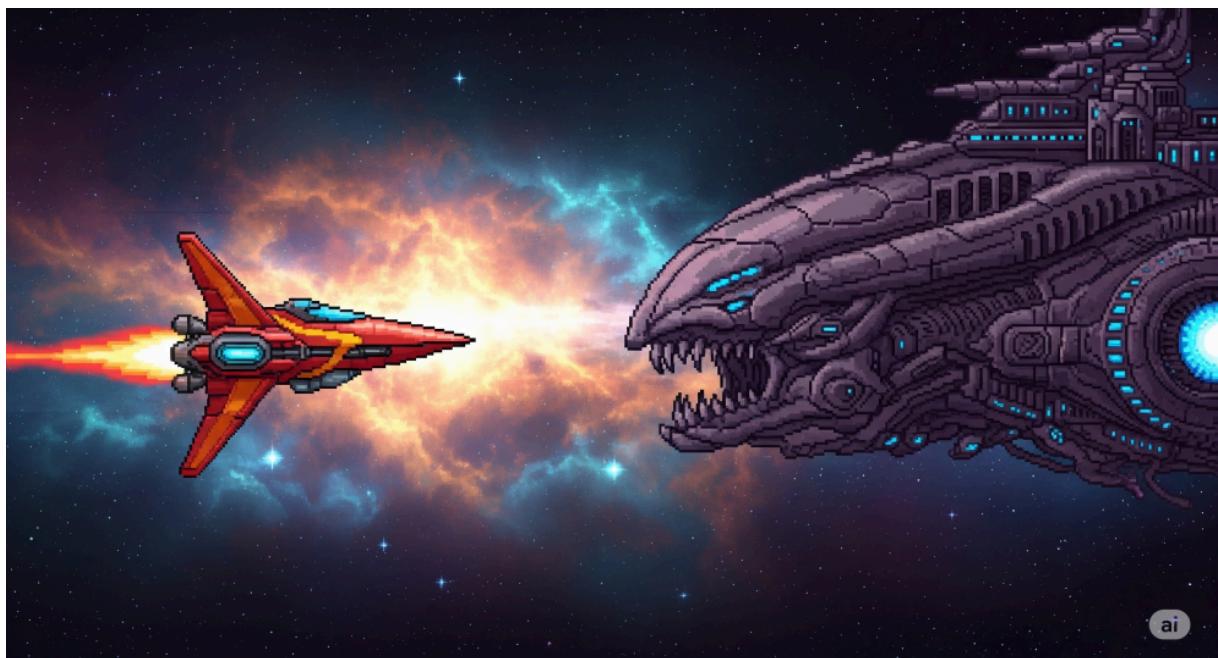
5.2 Paleta de Cores

O sistema utiliza as seguintes cores principais:

- **Sombra:** al_map_rgb(0, 0, 0) - Preto
- **Pontuação:** al_map_rgb(41, 74, 183) - Azul
- **Texto geral:** al_map_rgb(255, 255, 255) - Branco

5.3 Desenho das Telas

5.3.1 Tela de Menu



Fonte: Imagem gerada por Gemini.

```

323 void draw_menu(ALLEGRO_BITMAP *tela_menu, ALLEGRO_FONT *font) {
324     al_draw_bitmap(tela_menu, 0, 0, 0);
325     al_draw_text(font, al_map_rgb(255, 255, 255), SCREEN_W / 2, SCREEN_H - 100,
326                 ALLEGRO_ALIGN_CENTER, "Pressione ENTER para comecar");
327 }

```

5.3.2 Tela de Fim de Jogo



Fonte: Imagem gerada por Gemini.

```

329 void draw_fim_de_jogo(ALLEGRO_BITMAP *tela_fim_de_jogo, Pontuacao pontuacao, ALLEGRO_FONT *font) {
330     al_draw_bitmap(tela_fim_de_jogo, 0, 0, 0);
331
332     // Título
333     al_draw_text(font, al_map_rgb(255, 255, 255), SCREEN_W / 2, SCREEN_H / 2 - 60,
334                 ALLEGRO_ALIGN_CENTER, "GAME OVER");
335
336     // Pontuação final
337     al_draw_textf(font, al_map_rgb(255, 255, 255), SCREEN_W / 2, SCREEN_H / 2 - 20,
338                 ALLEGRO_ALIGN_CENTER, "SCORE: %d", pontuacao.pontuacao_atual);
339
340     // Recorde
341     al_draw_textf(font, al_map_rgb(255, 255, 255), SCREEN_W / 2, SCREEN_H / 2 + 20,
342                 ALLEGRO_ALIGN_CENTER, "RECORD: %d", pontuacao.recorde);
343
344     // Instruções
345     al_draw_text(font, al_map_rgb(255, 255, 255), SCREEN_W / 2, SCREEN_H / 2 + 80,
346                 ALLEGRO_ALIGN_CENTER, "Pressione R para jogar novamente");
347     al_draw_text(font, al_map_rgb(255, 255, 255), SCREEN_W / 2, SCREEN_H / 2 + 110,
348                 ALLEGRO_ALIGN_CENTER, "Pressione ESC para sair");
349 }
350

```

6 REFERÊNCIA DA BIBLIOTECA ALLEGRO 5

6.1 Funções de Inicialização

Função	Descrição
<code>al_init()</code>	Inicialização principal da biblioteca
<code>al_init_primitives_addon()</code>	Addon para formas geométricas
<code>al_init_image_addon()</code>	Addon para carregamento de imagens
<code>al_init_font_addon()</code>	Addon para sistema de fontes
<code>al_init_ttf_addon()</code>	Addon para fontes TrueType
<code>al_init_acodec_addon()</code>	Addon para codecs de áudio

6.2 Gerenciamento de Display

Função	Descrição
<code>al_create_display()</code>	Criação da janela de jogo
<code>al_destroy_display()</code>	Destruição da janela
<code>al_flip_display()</code>	Atualização da tela

6.3 Manipulação de Bitmaps

Função	Descrição
<code>al_load_bitmap()</code>	Carregamento de imagens
<code>al_draw_bitmap()</code>	Desenho de bitmap
<code>al_draw_scaled_bitmap()</code>	Desenho redimensionado
<code>al_destroy_bitmap()</code>	Liberação de memória

6.4 Sistema de Eventos

Função	Descrição
<code>al_create_event_queue()</code>	Criação da fila de eventos

<code>al_register_event_source()</code>	Registro de fonte de eventos
<code>al_wait_for_event()</code>	Espera por evento
<code>al_is_event_queue_empty()</code>	Verificação de fila vazia

6.5 Códigos de Eventos Principais

Constante	Descrição
<code>ALLEGRO_EVENT_TIMER</code>	Evento de timer (60 FPS)
<code>ALLEGRO_EVENT_KEY_DOWN</code>	Tecla pressionada
<code>ALLEGRO_EVENT_KEY_UP</code>	Tecla liberada
<code>ALLEGRO_EVENT_DISPLAY_CLOSE</code>	Fechamento da janela

7. CONCLUSÃO

A implementação do jogo Space Invaders em linguagem C utilizando a biblioteca Allegro 5 demonstra a aplicação prática de conceitos fundamentais de programação estruturada, incluindo gerenciamento de memória, manipulação de estruturas de dados, algoritmos de detecção de colisão e programação orientada a eventos.

O projeto aborda aspectos essenciais do desenvolvimento de jogos 2D, desde a inicialização do ambiente gráfico até a implementação de mecânicas complexas como movimento em grupo, sistema de pontuação e persistência de dados. A documentação apresentada fornece uma base sólida para compreensão e extensão do código, seguindo padrões de documentação técnica estabelecidos.

7.1 Contribuições Técnicas

- Implementação de sistema de detecção de colisões AABB eficiente
- Desenvolvimento de arquitetura modular para fácil manutenção
- Aplicação de boas práticas de programação em C
- Integração completa com biblioteca gráfica Allegro 5

7.2 Trabalhos Futuros

Possíveis extensões incluem implementação de múltiplos níveis, sistema de power-ups, inteligência artificial aprimorada para aliens e otimizações de performance para dispositivos com recursos limitados.

8. REFERÊNCIAS

ALLEGRO DEVELOPMENT TEAM. **Allegro 5 Reference Manual**. Disponível em: <https://liballeg.org/a5docs/trunk/>. Acesso em: 2024.

PIXABAY. **Músicas Hypernova para jogos**. Disponível em: <https://pixabay.com/pt/music/search/hypernova/?genre=jogos+de+v%C3%ADdeo+game>. Acesso em: 24 jun. 2025.

MIXKIT. **Músicas grátis para vídeo game**. Disponível em: <https://mixkit.co/free-stock-music/tag/video-game/>. Acesso em: 25 out. 2025.

OLMO, Pedro. Canal do YouTube. YouTube, 2006. Disponível em: <https://www.youtube.com/@pedroolmo>. Acesso em: 20 jun. 2025.