Documentação
Biblioteca e Funções

Integrantes:

- Alexandre Vasconcelos Vicente
- Artur Avallus Cordeiro de Paula
- João Pedro Sirqueira Costa
- Kauã Melo dos Santos
- Luiz Henrique Gonçalves e Souza
- Vinicius Andrade Silva Rodrigues

Link do Repositório Git:

https://github.com/JoaoPedroDevOz/Spiderman-game-allegro

Link do Vídeo:

https://youtu.be/neONy34PbUI

. Introdução

O nosso jogo foi feito utilizando do personagem de quadrinhos Homem Aranha como player principal. Dentro da aplicação, o jogador que controla o personagem pode ter a funcionalidade no mapa de pular e conseguir coletar moedas para atingir uma pontuação até o final do jogo. O objetivo do jogo é tentar coletar o máximo de moedas possível até o final da fase.

. Biblioteca Allegro

Allegro 5 é uma biblioteca multiplataforma voltada para o desenvolvimento de jogos 2D.

Ela facilita o uso de:

- Janelas e gráficos
- Imagens (bitmaps)
- Entrada do teclado e mouse
- Temporização (FPS e timers)
- Áudio
- Fontes e textos
- Colisões simples

É amplamente usada em jogos simples e projetos acadêmicos, justamente porque é leve, rápida e relativamente fácil de usar.

O funcionamento básico de um jogo utilizando Allegro envolve:

Inicialização da biblioteca

Chamadas como al_init(), al_install_keyboard(), etc.

Criação dos objetos essenciais

- ALLEGRO_DISPLAY → janela do jogo
- ALLEGRO_EVENT_QUEUE → fila de eventos
- ALLEGRO_TIMER → controla FPS
- ALLEGRO_BITMAP → imagens
- ALLEGRO_FONT → textos e menus

Loop principal do jogo

Esse loop:

- recebe eventos (teclas, timer, fechar janela)
- atualiza estados do jogo
- desenha a tela

Esse é o coração do jogo.

Encerramento

Libera memória com funções como al_destroy_bitmap, al_destroy_display, etc.

. Compilação

Siga as instruções abaixo para configurar o ambiente de desenvolvimento e executar o jogo:

Requisitos

- Sistema Operacional: Windows, Linux ou macOs
- Compilador C/C++ (GCC, Clang, MSVC)
- Biblioteca Allegro 5 instalada em seu sistema
- CMake (opcional, mas recomendado)
- Git (para trabalhar no repositório)
- Editor de código ou IDE (Visual Studio, Visual Studio Code, Code ::Blocks, etc.)
- Ferramenta de construção (Make, Ninja, etc.)
- Bibliotecas adicionais (se necessário, como SDL2, OpenGL, etc.)

Configuração do Ambiente

1. **Instalar a Biblioteca Allegro**
    - Entre no seguinte link dentro do GitHub para instalar a biblioteca.
    - Vá até “Releases” no canto direito da tela -> role o scroll do mouse até achar “Assets” -> procure pelo arquivo allegro-x86_64-w64-mingw32-gcc-15.2.0-posix-seh-static-5. para baixar.
    - Descompacte o arquivo.zip e salve a pasta Allegro no Disco Local C.
2. **Configurar o Projeto (pelo GCC)**
    - Baixe o arquivo zip do nosso projeto no repositório do GitHub ou no próprio AVA.
    - Acesse a pasta que está localizado o projeto dentro do Prompt de Comando (Cmd). Ex: cd Doc; cd Projeto.
    - Após isso, digite o seguinte comando: “**gcc -I C:\allegro\include -c main.c”**. Desta forma será criado o arquivo main.o. Digite o comando “dir” no cmd para verificar se o arquivo foi criado.
    - Após isso, copie os arquivos “liballegro_monolith.dll.a” dentro de alegro\lib e “allegro_monolith-5.2.dll” dentro de alegro\bin e coloque dentro da pasta do projeto. Ou se

preferir utilize o seguinte comando no próprio cmd para adicionar os arquivos: “**xcopy C:\allegro\bin\ allegro_monolith-5.2.dll**” e “**xcopy C:\allegro\lib\ liballegro_monolith.dll.a**”. *Obs: Pode substituir os arquivos que já estão na pasta caso peça.

- Agora, utilize o seguinte comando para criar um arquivo.exe do projeto: “**gcc -I C:\allegro\include main.o -o program.exe liballegro_monolith.dll.a”.**
- Por fim, basta rodar o jogo com o comando no cmd: “**program.exe”.**

*Obs: toda essa parte pra compilar está dentro do nosso repositório no GitHub também, basta acessar aqui. Dentro do repositório contém todos os arquivos do projeto, além do README.md.

Módulos da Allegro usados no projeto

Nosso projeto utiliza vários módulos diferentes da Allegro:

• allegro5/allegro.h

Funções principais da biblioteca: display, timers, eventos, núcleo do Allegro.

• allegro5/allegro_image.h

Permite carregar imagens como PNG, JPG, etc.

• allegro5/allegro_font.h

Manipula fontes básicas.

• allegro5/allegro_ttf.h

Carrega fontes TrueType (TTF), como Arial.ttf.

• allegro5/allegro_primitives.h

Permite desenhar linhas, quadrados, geometria simples (caso você use futuramente).

Principais funções da Allegro utilizadas no código

A seguir está uma lista com explicações rápidas e fáceis de entender.

Inicialização

al_init()

Inicializa o Allegro.
Sem isso, nada funciona.

al_install_keyboard()

Habilita entrada do teclado.

al_init_image_addon()

Permite carregar imagens externas (bitmaps).

al_init_font_addon()

Inicializa o sistema básico de fontes.

al_init_ttf_addon()

Permite usar fontes TTF, como Arial.ttf.

Display e eventos

al_create_display(largura, altura)

Cria a janela do jogo.

al_set_window_title()

Define o título da janela.

al_get_display_event_source(display)

Obtém eventos relacionados à janela (ex: clicar no X).

al_create_event_queue()

Cria a fila de eventos do jogo.

al_register_event_source(queue, source)

Registra o que envia eventos (teclado, display, timer).

al_get_next_event(queue, &event)

Obtém o próximo evento que ocorreu:

- tecla pressionada
- tecla solta
- timer
- fechar janela

Timers

al_create_timer(1.0 / 60.0)

Cria um timer para atualizar 60 vezes por segundo (60 FPS).

al_start_timer(timer)

Inicia o timer.

al_get_timer_event_source(timer)

Permite registrar o timer na fila de eventos.

Bitmaps (Imagens)

al_load_bitmap("caminho")

Carrega uma imagem e devolve um ponteiro tipo ALLEGRO_BITMAP.

Usado para carregar:

- personagem
- fundo
- tiles
- moedas
- objetos

al_draw_bitmap(img, x, y, flags)

Desenha a imagem na tela na posição (x, y).

Fontes e textos

al_load_font("arial.ttf", tamanho, flags)

Carrega uma fonte personalizada.

Usado no menu e placar.

al_draw_text(fonte, cor, x, y, alinhamento, texto)

Desenha texto na tela.
Utilizado em:

- menu
- score
- placar

Renderização e tela

al_clear_to_color(al_map_rgb(r,g,b))

Limpa a tela para uma cor.

al_flip_display()

Atualiza o conteúdo exibido na janela.
É o “refresh” do frame.

Funções relacionadas ao jogo (colisão, mapa, moedas)

Aqui estão as funções que implementamos e que fazem parte da lógica do jogo.

---

tileSolido(int id)

Retorna se um tile é sólido ou não, usando uma lista de IDs.
É usado para colisão com o chão.

---

tileMoeda(int id)

Verifica se um tile é uma moeda.

---

checarColisaoComTiles()

Função importante do jogo.

Ela:

- pega a posição do personagem
- calcula quais tiles ele está tocando
- verifica colisão com o chão
- impede que o personagem atravesse o tile
- reseta velocidade vertical quando encosta no solo
- controla se ele está pulando ou não

Usamos:

- altura do sprite
- offset horizontal do mapa
- função tileSolido

---

coletarMoedas()

Verifica se o personagem está encostando em algum tile de moeda.

Quando está:

- remove a moeda (define tile como 0)
- aumenta moedas_coletadas

Utiliza bounding box simples.

---

gerarMoedasAleatoriasTempoReal()

Gera moedas dinamicamente no mapa.

Ela:

- calcula posição futura do jogador
- escolhe uma coluna à frente
- procura um chão sólido
- coloca moeda acima dele

Isso dá sensação de mapa infinito.

---

renderizarMapaRepetindo()

Desenha o mapa como um “loop infinito”.

O mapa é repetido centenas de vezes:

while (copia != 0)

Cada cópia:

- calcula posição X com offset
- desenha os tiles
- gera moedas

Quando o mapa sai da tela:

tile_offset_x += larguraMapa;

O jogador vê um "mundo infinito".

Estruturas importantes

struct PersonagemBase

Guarda atributos gerais do personagem:

- estado do pulo
- tempo do pulo
- duração do pulo
- velocidade

struct Spiderman

Guarda:

- posição
- velocidade vertical
- largura/altura
- animação (frame)
- linha atual do sprite
- PersonagemBase

Controle do jogador

O código usa:

ALLEGRO_EVENT_KEY_DOWN

Quando tecla é pressionada.
Usado para:

- pular
- navegar menu

ALLEGRO_EVENT_KEY_UP

Evita repetição da tecla (key repeat).

Estados do jogo (enum GameState)

MENU
PLAYING

PLACAR

Utilizamos um **switch implícito**, alternando o comportamento:

- Se está no MENU → desenha e controla menu
- Se está PLAYING → roda o jogo
- Se está PLACAR → exibe pontuação

Esse é um padrão clássico em jogos 2D.

Mapa do Jogo

Na lógica do mapa, utilizamos uma matriz para preencher com blocos (tiles) e objetos (objects) na tela. Basicamente, colocamos números nessa matriz para dizer qual a identificação do bloco. Ex: 1 – escada; 2 – piso; etc.

```
// --- Mapa do jogo ---
int mapa[MAPA_LINHAS][MAPA_COLUNAS] = {
    {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3},
    {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,13,13,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,13,13,13,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,11,11,11,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,13,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,13,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,11,11,11,11,11},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,0,0,13,0,0,0,0,0,0,0,0,0},
    {0,0,4,0,0,0,0,0,0,0,0,0,13,13,0,0,0,0,0,0,0,0,0,0,0,0,6,6,6,6,6,6,0,0,0,0,0,0,0,0},
    {0,0,4,0,0,0,0,0,0,0,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,5,0,0,0},
    {0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,5,5,5,0},
    {0,7,4,0,0,0,0,12,0,0,0,13,13,13,13,0,0,0,0,0,0,0,0,0,13,0,8,9,9,9,9,10,0,13,0,5,5,5,5,5},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3},
    {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3},
    {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3},
};
```

Desta forma, para o programa reconhecer cada bloco, fizemos um vetor referenciando a struct ALLEGRO_BITMAP*, inserindo em cada

índice dele um bloco. Portanto, já foi inserida uma função da biblioteca Allegro junto para desenhar o bloco no mapa.

```
120
121         ALLEGRO_BITMAP* tiles[14] = { NULL };
122         tiles[1] = al_load_bitmap("./img/Tiles/piso1.png");
123         tiles[2] = al_load_bitmap("./img/Tiles/plataforma3.png");
124         tiles[3] = al_load_bitmap("./img/Tiles/piso5.png");
125         tiles[4] = al_load_bitmap("./img/Objects/escada.png");
126         tiles[5] = al_load_bitmap("./img/Tiles/bloco.png");
127         tiles[6] = al_load_bitmap("./img/Tiles/plataforma1.png");
128         tiles[7] = al_load_bitmap("./img/Objects/placa.png");
129         tiles[8] = al_load_bitmap("./img/Objects/barra1.png");
130         tiles[9] = al_load_bitmap("./img/Objects/barra2.png");
131         tiles[10] = al_load_bitmap("./img/Objects/barra3.png");
132         tiles[11] = al_load_bitmap("./img/Tiles/plataforma2.png");
133         tiles[12] = al_load_bitmap("./img/Objects/caixa.png");
134         tiles[13] = al_load_bitmap("./img/Objects/coin3.png");
135
```

Após isso, realizamos um loop para ficar gerando o mapa repetidas vezes para dar a impressão de que o mapa está andando.

```
137         }
138     }
139
140     void renderizarMapaRepetindo(
141         int mapa[][40],
142         ALLEGRO_BITMAP* tiles[],
143         float tile_offset_x,
144         int linhas,
145         int colunas,
146         int largura_tela,
147         int altura_tela,
148         int TILE,
149         int posicao_bg_x,
150         float pos_x_personagem
151     ) {
152         int larguraMapa = colunas * TILE;
153
154         int copia = 100;
155         while (copia != 0) {
156             copia--;
157             float offset = tile_offset_x + copia * larguraMapa;
158
159             if (offset + larguraMapa > -TILE && offset < largura_tela + TILE) {
160                 gerarMoedasAleatoriasTempoReal(mapa, offset, TILE, largura_tela, copia, pos_x_personagem);
161             }
162
163             for (int linha = 0; linha < linhas; linha++) {
164                 for (int coluna = 0; coluna < colunas; coluna++) {
165
166                     int t = mapa[linha][coluna];
167                     if (t == 0) continue;
168
169                     float draw_x = coluna * TILE + offset;
170                     float draw_y = linha * TILE;
171
172                     if (draw_x > -TILE && draw_x < largura_tela)
173                         al_draw_bitmap(tiles[t], draw_x, draw_y, 0);
174                 }
175             }
176         }
```

Além disso, todo o background está “andando” para a esquerda para dar a impressão de movimento. Assim, o personagem fica praticamente parado no mesmo lugar com uma animação sendo trocada, além de ter a capacidade de pular.

Referências Utilizadas

- Documentação da biblioteca Allegro: https://liballeg.org/

- Chat GPT: utilizamos para tirar dúvidas e nos dar ideias do que fazer para resolver certos problemas que estávamos enfrentando.

- Vídeos no Youtube:

https://youtu.be/mJFYV8Hk6jY?si=l1tC1b5b_EpN4b_O

https://youtu.be/F8_VFb7Vyvo?si=rFDO0Xw2ONrabufv

https://youtu.be/jGgJuw9ibL4?si=pKtNpyT-upy9ydBS

- Assets utilizados (Tiles, Objects e Background):

https://craftpix.net/freebies/free-seaport-tileset-32x32-pixel-art-for-platformer/?num=1&count=79&sq=map%20city&pos=11

Pasta: Industrial Zone