



A Mimir

do planejamento à versão final

Objetivo do jogo

Desenvolvimento de um jogo estilo *swarm* em pixel art com a temática voltada ao sono e suas etapas. O personagem deve sobreviver enfrentando os inimigos com o objetivo de alcançar uma boa noite de sono.

Funcionamento idealizado

- O jogo tem um total de 5min de duração em que cada minuto simboliza uma fase do ciclo do sono
- O personagem pode fugir dos inimigos (cafeína e ansiedade) ou atirar projéteis “Zs” para eliminá-los
- Ao eliminar os inimigos, acumula XP que o permite subir de nível e desbloquear upgrades



Sketch inicial do jogo



Jogo implementado



Menu Final



Level Up!

A mimir v0.6 - HP: 5/10 | Pontos: 70

HP: 5/10 | PONTOS: 70

4:49

XP: 0 / 15

--- LEVEL UP! ---

Escolha um upgrade (1, 2 ou 3):

1. Velocidade de Movimento +1
2. Dano Base +10%
3. Cadencia de Tiro +10%



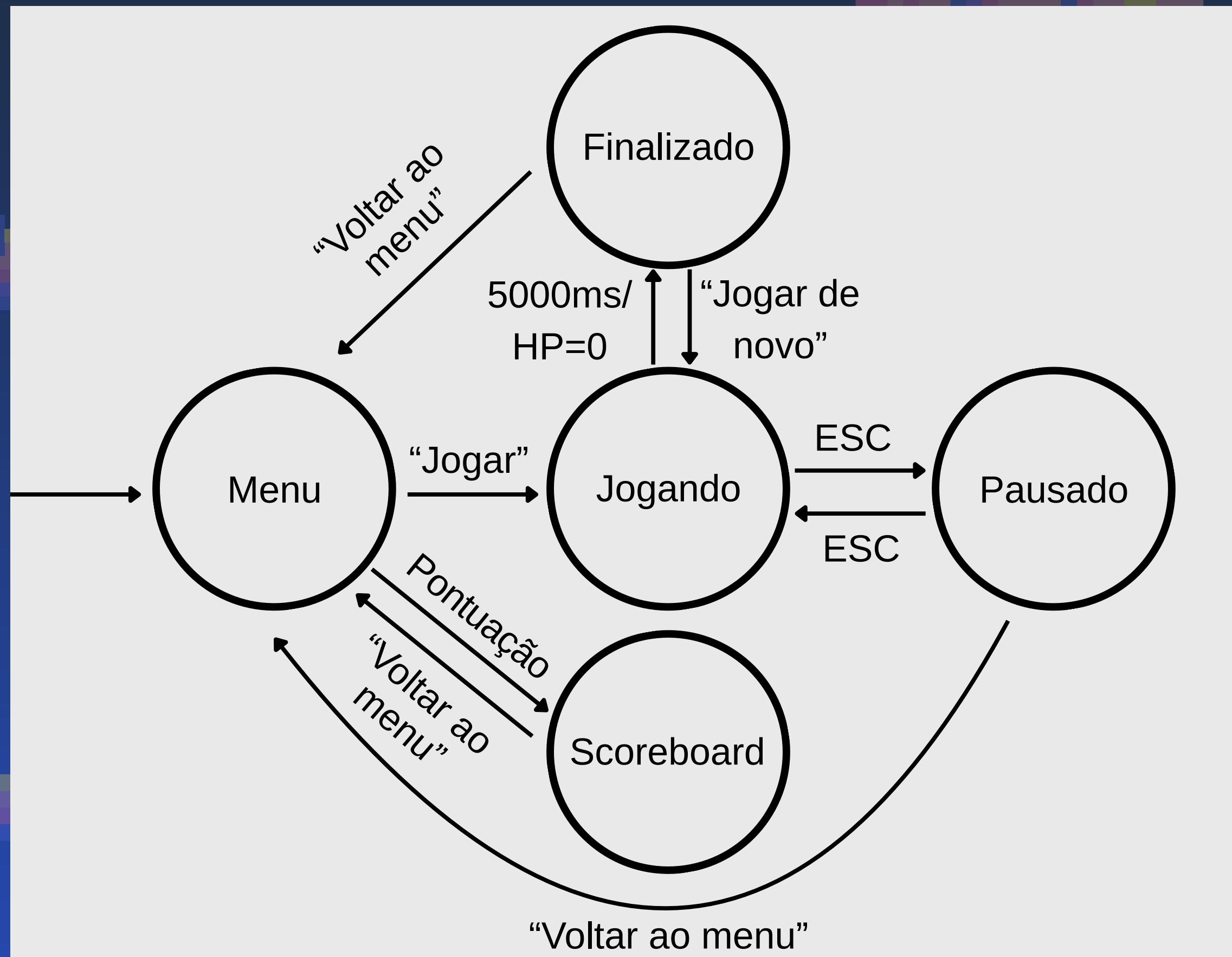
Tela de Fim de Jogo



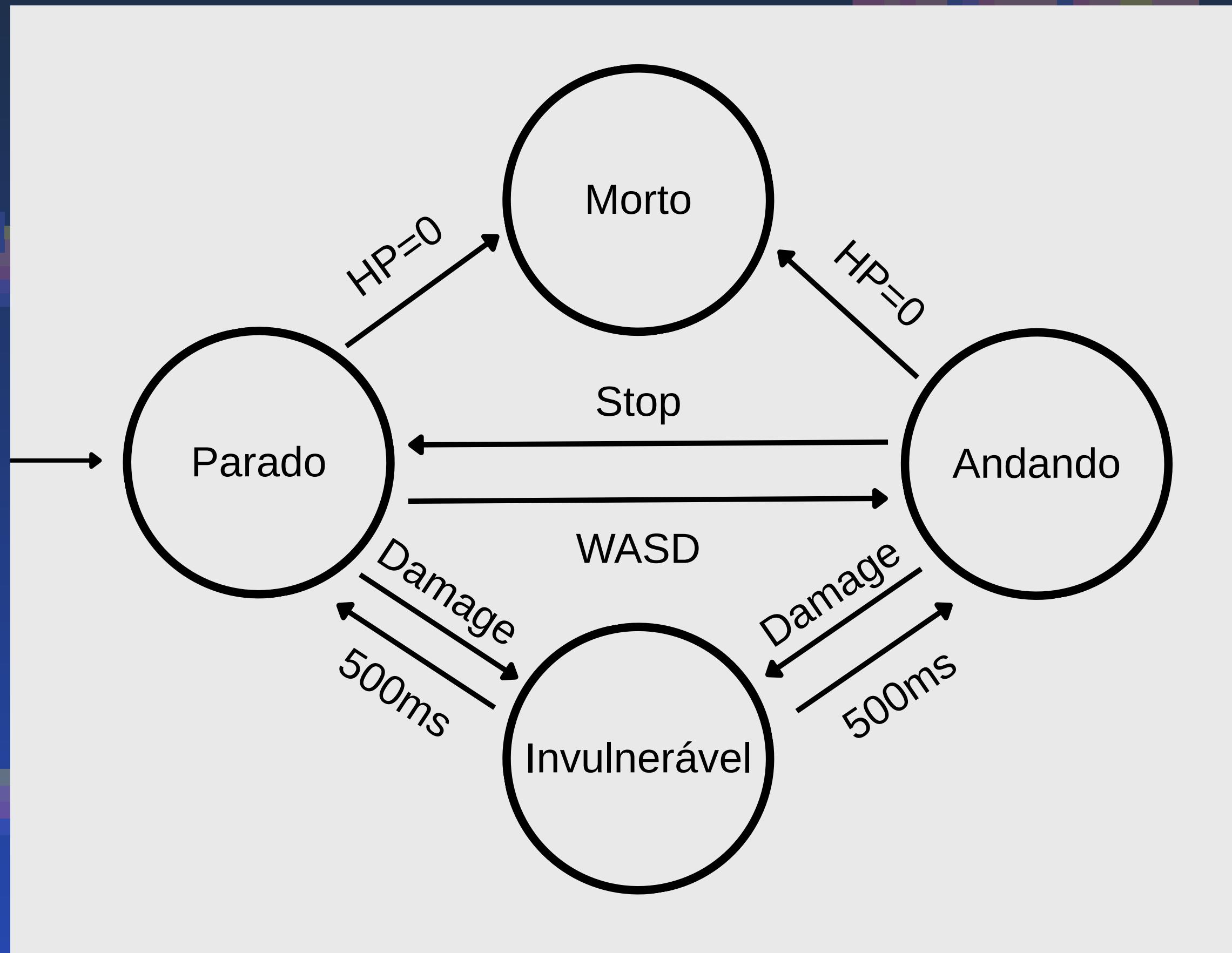
Arquitetura Geral do Jogo

```
1 # --- Variáveis de Configuração ---
2
3 # O compilador que queremos usar
4 CC = gcc
5
6 # Nome do seu programa executável final
7 TARGET = amimir
8
9 # Flags (opções) para o compilador:
10 # -Wall -Wextra = Mostra todos os avisos (MUITO recomendado)
11 # -g = Inclui informações de debug (para usar com o gdb)
12 # -std=c99 = Usa o padrão C99
13 CFLAGS = -Wall -Wextra -g -std=c99 -I/usr/include/SDL2
14
15 # Bibliotecas (libs) que precisamos linkar:
16 # -lSDL2 = A biblioteca principal da SDL2
17 # -lm = A biblioteca de matemática (para sqrtf, etc.)
18 LIBS = -lSDL2 -lSDL2_ttf -lm -lSDL2_image
19
20 # --- Arquivos do Projeto ---
21
22 # Lista de todos os seus arquivos-fonte (.c)
23 SRC = main.c init.c input.c update.c render.c text.c menu.c fim.c
24
```

FSM Geral



FSM Player



defs.h

```
// iluminação  
int currentDarknessAlpha;  
SDL_Texture *texDarknessLayer; // camada preta que cobre a tela  
SDL_Texture *texSpotlight; // "forma" da luz (circulo)  
  
// timer  
Uint32 roundStartTime;  
Uint32 elapsedTime;  
  
// Sistema de Upgrade  
UpgradeType currentUpgradeOptions[3];  
  
} App;  
  
// --- Tipos de Upgrade ---  
typedef enum {  
    UPGRADE_HP_MAX,  
    UPGRADE_FIRE_RATE,  
    UPGRADE_MOVE_SPEED,  
    UPGRADE_DAMAGE,  
    UPGRADE_MULTISHOT,  
    UPGRADE_BULLET_SPEED,  
    UPGRADE_DODGE,  
    UPGRADE_COUNT  
} UpgradeType;
```

upgrades aleatórios (main.c)

```
// sorteia 3 upgrades diferentes
int picked[3] = {-1, -1, -1};
for (int i = 0; i < 3; i++) {
    bool unique;
    int r;
    do {
        unique = true;
        r = rand() % UPGRADE_COUNT;
        for (int j = 0; j < i; j++) {
            if (picked[j] == r) unique = false;
        }
    } while (!unique);
    picked[i] = r;
    app.currentUpgradeOptions[i] = (UpgradeType)r;
}
printf("Level Up! Opções geradas.\n");
```

spotlight (render.c)

```
if (app->texDarknessLayer != NULL && app->texSpotlight != NULL) {  
  
    // Define a texture de luz como o ALVO  
    SDL_SetRenderTarget(renderer, app->texDarknessLayer);  
  
    // Calcula a luz ambiente baseada no timer  
  
    int ambientLight = 255 - app->currentDarknessAlpha;  
    if (ambientLight < 20) ambientLight = 20; // Limite mínimo de escuridão total  
  
    // Limpa a tela com a cor do ambiente (Cinza Escuro se for noite, Branco se for dia)  
    SDL_SetRenderDrawColor(renderer, ambientLight, ambientLight, ambientLight, 255);  
    SDL_RenderClear(renderer);
```

spotlight (render.c)

```
// Desenha a "Lanterna" (soma luz branca) onde o player está
SDL_Rect lightRect;

lightRect.w = SPOTLIGHT_RADIUS * 2;
lightRect.h = SPOTLIGHT_RADIUS * 2;
lightRect.x = player.rect.x + (player.rect.w / 2) - (lightRect.w / 2);
lightRect.y = player.rect.y + (player.rect.h / 2) - (lightRect.h / 2);

// Desenha a luz apagando a escuridão
SDL_RenderCopy(renderer, app->texSpotlight, NULL, &lightRect);

// Reseta o ALVO para a tela padrão
SDL_SetRenderTarget(renderer, NULL);

// Desenha a camada de escuridão pronta por cima do jogo
SDL_RenderCopy(renderer, app->texDarknessLayer, NULL, NULL);

}
```

spotlight (update.c)

```
//Calcula o número de degraus (steps) de escuridão concluidos
int completedSteps = app->elapsedTime / DARKNESS_STEP_MS;

float alphaPerStep = (float)MAX_MAX_DARKNESS_ALPHA / (ROUND_DURATION_MS / DARKNESS_STEP_MS);

int targetAlpha = (int)(completedSteps * alphaPerStep);

if (targetAlpha > MAX_MAX_DARKNESS_ALPHA) {
    targetAlpha = MAX_MAX_DARKNESS_ALPHA;
}

app->currentDarknessAlpha = targetAlpha;

if (app->elapsedTime >= ROUND_DURATION_MS) {
    // FIM DE JOGO
    *currentState = STATE_END;
    return;
}
```

‘waves’ (update.c)

```
// LÓGICA DE DIFÍCULDADE BASEADA NO TEMPO ---  
  
// Exemplo: Aumentar spawn rate na metade do tempo (2.5 minutos)  
Uint32 timeLimitThreshold = ROUND_DURATION_MS / 2;  
int currentSpawnInterval = ENEMY_SPAWN_INTERVAL; // 800ms  
  
if (app->elapsedTime > timeLimitThreshold) {  
    currentSpawnInterval = ENEMY_SPAWN_INTERVAL / 2; // 400ms - Dobra a velocidade de spawn  
    // mudar os tipos de inimigo aqui  
}
```

Conclusão

- O que foi aprendido
 - Modularização
- Pontos fortes do projeto
 - Escalabilidade
 - Rejogabilidade
- Principais dificuldades
 - Gestão de Estados
 - Sistema de Level Up
- Ideias de possíveis melhorias
 - Áudio
 - Scoreboard