

RELATORIO:

Trabalho Final Pac-Man Prog 2 - UFRJ

Alunos:

- 1 - Matheus de Castro Dell Orto Azeredo - (123110312)
- 2 - Rode Kong - (Dre: 122116284)
- 3 - Lucas Rodrigues Araujo - (Dre: 123134439)
- 4 - Gabriel Delgado Aguiar Moreira - (Dre: 120033804)

Seção com uma descrição do trabalho (quais funcionalidades foram implementadas):

No projeto Pac-Man, as funcionalidades implementadas incluem:

- Controle do Pac-Man: Movimentação do personagem principal pelo mapa.
- Fantasmas: Implementação dos inimigos com lógica de perseguição.
- Sistema de colisão: Detecção de colisões entre Pac-Man, fantasmas e elementos do mapa.
- Gerenciamento de mapas: Carregamento e integração dos mapas com a tela principal.
- Interface gráfica: Exibição visual do jogo.
- Menu inicial: Navegação e início da partida.

Essas funcionalidades estão organizadas em arquivos específicos do projeto:

- entidades.c/.h: Gerenciam os personagens (Pac-Man e fantasmas).
- colisoes.c/.h: Implementam a lógica de colisão.
- mapa.c/.h: Manipulam o carregamento e atualização dos mapas.
- graficos.c/.h: Cuidam da renderização gráfica (parte visual).
- menu.c/.h: Controlam o menu inicial e navegação.
- main.c: Integra todas as partes e gerencia o fluxo principal do jogo.

Seção com as decisões do trabalho (como funcionalidades foram implementadas);

Um resumo detalhado de como as funcionalidades foram implementadas em cada arquivo .c presente no projeto:

Durante o desenvolvimento do projeto Pac-Man, diversas decisões técnicas foram tomadas para garantir a funcionalidade, desempenho e organização do código. Abaixo estão as principais escolhas e justificativas:

- Estrutura Modular:

O projeto foi dividido em múltiplos arquivos (main.c, jogo.c, mapa.c, entidades.c, etc.), cada um responsável por um aspecto específico do jogo. Essa abordagem facilita a manutenção e o entendimento do código.

- Gestão de Entidades:

As entidades do jogo (Pac-Man, fantasmas, frutas) foram implementadas em entidades.c e entidades.h, utilizando structs para representar seus estados e funções para manipulação. Isso permitiu um controle eficiente sobre movimentação e colisões.

- Sistema de Colisões:

As colisões entre entidades e o mapa foram tratadas em colisoes.c, separando a lógica de detecção e resposta a colisões. Isso tornou o código mais limpo e permitiu fácil ajuste das regras de

interação.

- Renderização Gráfica

A exibição dos elementos visuais foi centralizada em graficos.c, utilizando funções para desenhar o mapa, entidades e interface. Essa separação garantiu que mudanças visuais não afetassem a lógica do jogo.

- Gestão do Mapa

O mapa do jogo foi implementado em mapa.c, com funções para carregar, modificar e exibir o labirinto. A escolha por um array bidimensional facilitou a verificação de posições e movimentação dos personagens.

- Menu e Interface

O menu principal e as opções de jogo foram desenvolvidos em menu.c, permitindo navegação intuitiva e separando a lógica de interface da lógica do jogo.

- Makefile

A compilação do projeto foi automatizada via makefile, simplificando o processo de build e garantindo que todas as dependências fossem corretamente gerenciadas.

Seção com a divisão das responsabilidades (quem implementou as funcionalidades);

- Parte das colisões:(Lucas)

- colisoes.h

//tipos de dados:

enum TipoColisao -> define os 7 tipos possíveis de interação (PACMAN_FANTASMA, FANTASMA_PAREDE, FANTASMA_FANTASMA, PORTAL).

struct ResultadoColisao -> estrutura de resposta que contém informações sobre o resultado de uma colisão (tipo de colisão, entidades envolvidas, index do inimigo, etc).

//Protótipos de Funções (API):

verificar_colisao_pacman -> sensor principal para retornar o ResultadoColisao baseado na próxima posição do jogador.

verificar_colisao_fantasma_mapa -> validação de terreno para IA, impede a entrada em paredes.

verificar_colisao_fantasma_fantasma -> controle de sobreposição, impede que dois fantasmas ocupem a mesma coordenada.

processar_portal -> calcula as coordenadas de destino ao entrar em um túnel.

verificarTodasColisões -> aplica as regras do jogo, alterando o estado global.

colisoes.C:

função: contar_fantasmas_lista -> percorre a lista de entidades e conta quantas são fantasmas.

função: verificar_colisao_pacman -> verifica se o pacman colidiu com algum fantasma na lista de entidades.

função: verificar_colisao_fantasma_mapa -> verifica se algum fantasma colidiu com as paredes do mapa.

função: verificar_colisao_fantasma_fantasma -> verifica se algum fantasma colidiu com outro fantasma na lista de entidades.

função: processar_portal -> executa a mecânica de teletransporte quando o pacman ou um fantasma entra em um portal.

função: verificarTodasColisões -> função principal que chama todas as funções de verificação de

colisões e processa as consequências dessas colisões.
altera o estado global do jogo conforme necessário (ex: reduzir vidas do pacman, reiniciar posições, etc).

- Parte dos Mapas e Entidades:(Matheus)
- mapa.h - Cabeçalho do sistema de mapas

Definição da estrutura Mapa: Armazena matriz do labirinto, dimensões, pellets restantes, nome do arquivo e posições dos fantasmas

Funções principais:

carregarMapa(): Carrega um mapa de arquivo

liberarMapa(): Libera memória do mapa

contarPontosMapa(): Retorna número de pellets restantes

verificarCelula(): Verifica tipo de célula em posição específica

definirCelula(): Modifica valor de uma célula

remover_pellet() e remover_power_pellet(): Remove pellets específicos

mapa.c - Implementação do sistema de mapas
Leitura de arquivos de mapa:

Suporte a diferentes formatos de quebra de linha (Windows/Linux)

Extração automática do nome do arquivo

Validação de dimensões

Processamento de células:

Identifica diferentes elementos: # (paredes), . (pellets), O (power pellets), F (fantasmas), T (portais)

Contagem automática de pellets e power pellets

Extração de posições dos fantasmas do arquivo

Gerenciamento de memória:

Alocação dinâmica da matriz

Limpeza adequada de recursos

Funcionalidades adicionais:

Suporte a teleporte horizontal (túneis)

Debug com impressão de informações do mapa

- entidades.h - Cabeçalho do sistema de entidades

Definições de estruturas:

Posicao: Coordenadas (linha, coluna)

PacMan: Jogador com posição, vidas, pontos, timer de power-up e direção

Fantasma: Inimigos com posição, estado de vulnerabilidade, timer, direção e contador de frames individual

Sistema de direções: Constantes para movimentação (cima, baixo, esquerda, direita)

Funções de gerenciamento:

Criação e destruição de entidades

Movimentação com colisão

IA de fantasmas (perseguição/fuga)

Controle de velocidade

Atualização de estados

Variáveis globais: Ponteiros para Pac-Man e lista de fantasmas

- entidades.c - Implementação do sistema de entidades
Sistema de velocidade independente:

Contadores de frames individuais para Pac-Man e cada fantasma

Fatores de velocidade ajustáveis (1-5)

Funções para aumentar/diminuir velocidade

Inicialização inteligente:

Posicionamento automático do Pac-Man em local válido

Criação de fantasmas a partir das posições do mapa

Verificação de colisões iniciais

Movimentação avançada:

Pac-Man: Movimento controlado por input, coleta de pellets, ativação de power-ups

Fantasmas:

Modo perseguição: Minimiza distância ao Pac-Man

Modo fuga (vulnerável): Maximiza distância do Pac-Man

Evita oscilação (não volta pela mesma direção)

Teleporte através de túneis

Sistema de colisão:

Colisão com paredes

Colisão Pac-Man x Fantasma com diferentes resultados baseados no estado

Coleta de pellets e power pellets

Estados especiais:

Power pellet: Torna fantasmas vulneráveis por tempo limitado

Timer de vulnerabilidade com efeitos visuais

Gerenciamento de jogo:

Sistema de vidas

Pontuação (10 pontos por pellet, 50 por power pellet, 200 por fantasma comido)

Reposicionamento após morte

Finalização de jogo

Debug: Funções para imprimir estado completo das entidades

- Parte da Main e do menu:(Gabriel)

1. main.c: O Orquestrador

Este arquivo é o ponto de entrada (entry point) da aplicação. Sua função é extremamente enxuta, servindo apenas para inicializar e manter o ciclo de vida do jogo.

- Ciclo de Vida: Inicializa a janela e o contexto do jogo (inicializarJogo), entra no loop principal do Raylib (while !WindowShouldClose) e, ao final, limpa a memória.
- Delegação: Ele não contém lógica de jogo; apenas chama funções de atualização (atualizarJogo) e desenho (desenharJogo) definidas em outros módulos.

2. menu.h: As Definições (Interface)

Este cabeçalho define a estrutura de dados e os contratos para o sistema de menus.

- Máquina de Estados: Define o enum EstadoMenu para controlar onde o usuário está (Menu Principal, Seleção de Nível, Confirmar Saída).
- Estrutura de Dados: A struct Menu centraliza tudo: estado atual, animações, recursos gráficos (texturas/fontes) e dados do sistema de salvamento (temSave, saveFileName).
- Opções: Define as constantes para as opções do menu: Novo Jogo, Carregar e Sair.

3. menu.c: A Lógica do Menu

Este é o arquivo mais complexo, contendo a implementação visual e funcional do menu.

- Gerenciamento de Input:
- Usa WASD ou Setas para navegar entre as opções e Enter/Espaço para selecionar.
- Possui lógica para transitar entre estados (ex: ao escolher "Novo Jogo", muda o estado para MENU_SELECAO_NIVEL).
- Sistema de Save/Load (Persistência):
- Salvar: A função salvar_jogo grava dados binários (fwrite) contendo nível, pontos e vidas no arquivo savegame.dat.
- Carregar: A função carregar_jogo lê esses dados binários.
- Flag de Carregamento: Curiosamente, quando o jogador escolhe "Carregar Jogo", o código define menu->nivelSelecionado = -1 como um sinalizador especial para o motor do jogo saber que deve carregar um save em vez de iniciar um nível novo.
- Renderização e Visual:
- Define cores personalizadas (ex: COR_TITULO, COR_FUNDO_MENU).
- Desenha botões com animações simples (efeito de escala/seno quando selecionado) e carrega

fontes personalizadas (pacfont.ttf) se disponíveis.

- Parte dos Graficos: (Rode)
Graficos.c

Implementei a biblioteca (biblioteca padrão da linguagem C) e inclui Definições apresentadas no arquivo graficos.h

Funções que foram implementadas.

void inicializar_janela(void); Inicializa a janela do jogo com a resolução e FPS definidos

void fechar_janela(void); Fecha a janela do jogo e encerra o contexto graficos

void desenhar_mapa(Mapa* mapa); Desenha todo o mapa passado como argumento

void desenhar_pacman(PacMan* pacman) Foi alocado para desenhar o personagem Pac-Man

void desenhar_fantasmas(Fantasma*lista); Foi alocado para desenhar todos os fantasmas da lista alocada

void desenhar_hud(PacMan*pacman, Mapa*mapa, int nivel); Desenha a HUB com as informações(quantidade de vida, Pontuação, nível, pellets restantes)

int tecla_para_direção(void); Essa função foi implementada para converter/traduzir os comandos indicados no teclado para o código
(Key_W ou Key_up = cima, Key_s ou Key_down = baixo, Key_a ou Key_left = esquerda, Key_d ou Key_right = direita)

Graficos.h

Foi implementada as funções de alguns arquivos utilizados(mapa.h e entidades.h), e fiz uma definição de cores para cada item do jogo (parede,fantasma, personagens e etc), definimos também o tamanho da tela/hub de acordo com o relatório do projeto apresentado em aula
(Largura de Janela 1600 e a sua altura 840)