UFRJ

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

PROFESSOR: MARCOS TOMAZZOLI LEIPNITZ

ALUNOS: YASMYN DOS SANTOS PIRES, DRE: 122093826

GUSTAVO FELICIDADE DA COSTA, DRE: 118171109

PAC-MAN – PROGRAMAÇÃO DE COMPUTADORES II

Introdução

O presente relatório técnico tem por finalidade documentar o desenvolvimento do projeto "Pac-Man Prog II", uma implementação do clássico jogo de arcade Pac-Man, realizada no contexto da disciplina de Programação II. O projeto foi concebido com o objetivo de aplicar e consolidar os conhecimentos adquiridos em sala de aula, com foco na utilização de recursos avançados da linguagem C, tais como estruturas de dados complexas, manipulação de ponteiros, alocação dinâmica de memória e modularização do código.

A escolha da biblioteca gráfica Raylib permitiu a criação de uma interface gráfica funcional e o gerenciamento eficiente do loop principal do jogo, mantendo a implementação estritamente dentro do escopo de C padrão.

Este documento está estruturado para fornecer uma visão abrangente do projeto, detalhando a arquitetura de software, as decisões de design tomadas, a análise funcional de cada módulo de código e, de forma exaustiva, a divisão de responsabilidades e o fluxo de trabalho adotado pelos integrantes do grupo. O foco principal é demonstrar a aplicação prática dos conceitos de programação estruturada e a gestão de um estado de jogo complexo sem o uso de variáveis globais, garantindo a robustez e a manutenibilidade do código.

O projeto **'Pac-Man Prog II'** consiste na recriação do clássico jogo Pac-Man utilizando a linguagem C padrão e a biblioteca gráfica Raylib para renderização e gerenciamento de entrada. A implementação fundamenta-se na aplicação prática de conceitos essenciais da disciplina de **Programação II**, tais como modularização, alocação dinâmica de memória, manipulação de ponteiros e estruturas de dados complexas. A arquitetura do software prioriza a robustez e o encapsulamento, eliminando o uso de variáveis globais e centralizando o controle do jogo explicitamente na estrutura GameState.

1. Análise da Estrutura de Arquivos e Módulos

O projeto adota uma arquitetura modular, onde cada par de arquivos .h e .c é responsável por um domínio funcional específico do jogo.

1.1 Estrutura de diretórios

Diretório	Descrição
src/	Contém todo o código-fonte do jogo.
assets/maps	Armazena os arquivos de mapa em formato texto.

1.2 Detalhamento dos arquivos-fonte (src/)

Arquivo	Propósito Funcional	Estruturas/Funções Chave
main.c	Ponto de entrada e orquestração do loop principal.	main(), InitWindow(), WindowShouldClose(), CloseWindow().
game.h/.c	Gerenciamento centralizado do estado e lógica de jogo.	GameState struct, game_init(), game_update(), game_free().
map.h/.c	Manipulação, carregamento e liberação de mapas.	Map struct, map_load(), map_free().
entity.h	Definições de tipos e estruturas de entidades.	Position, Direction, Pacman struct, Ghost struct.

Arquivo	Propósito Funcional	Estruturas/Funções Chave
render.h/.c	Tradução do estado lógico para representação gráfica.	render_map(), render_entities(), render_hud().
menu.h/.c	Implementação da lógica e interface do menu de opções.	menu_update(), menu_draw().
save.h/.c	Serialização e desserialização do estado de jogo (persistência).	save_game(), load_game().

2. Decisões de projeto e justificativas técnicas

2.1 Gerenciamento de estado e modularidade

A arquitetura do projeto é fundamentada na centralização do estado do jogo na estrutura GameState. A passagem do estado por ponteiro (GameState *) para todas as funções de lógica e renderização é uma decisão de design crucial.

Justificativa: Esta metodologia garante a ausência de variáveis globais, promovendo um código com baixo acoplamento e alta rastreabilidade. O estado do jogo é sempre manipulado de forma explícita, o que facilita a depuração, a manutenção e a integração dos módulos.

2.2 Implementação do movimento discreto e controlado por tempo

O movimento das entidades é implementado de forma discreta (baseado em grade) e sincronizada por tempo, em vez de ser dependente da taxa de quadros (FPS).

- Mecanismo de moveTimer: O campo moveTimer presente nas estruturas de entidade é utilizado para controlar a cadência do movimento. A cada frame, o tempo decorrido (deltaTime) é subtraído do timer. A entidade só avança um bloco na grade quando o timer zera, garantindo uma taxa de movimento constante de 4 blocos por segundo.
- Direção Pendente (pendingDir): O Pac-Man utiliza a direção pendente para registrar a intenção do jogador. A mudança de direção só é efetivada quando o Pac-Man alcança o centro de um bloco e a nova direção é validada (ausência de parede).

Justificativa: O movimento discreto simplifica a lógica de colisão e a interação com os elementos da grade (pellets, portais), pois as coordenadas da entidade estão sempre

alinhadas com a matriz do mapa. O controle por tempo garante a consistência da jogabilidade em diferentes ambientes de execução

2.3 Lógica de fantasmas (inteligência artificial simples)

A IA dos fantasmas é baseada em um conjunto de regras de navegação em grade:

- Movimento contínuo: O fantasma mantém a direção atual até encontrar uma interseção ou uma parede.
- Seleção de direção: Em uma interseção (ponto com múltiplas saídas válidas), o fantasma seleciona uma nova direção de forma aleatória.
- Restrição de reversão: A direção oposta à direção atual é excluída das opções de movimento para evitar o comportamento de "ping-pong" e promover a exploração do mapa.

Justificativa: Esta lógica atinge um equilíbrio entre a complexidade de implementação e a imprevisibilidade necessária para o jogo, sem recorrer a algoritmos de busca de caminho mais complexos, como o A, que não seriam necessários para o escopo do projeto.*

2.4 Persistência de dados em formato binário

O módulo save.h/.c utiliza a serialização e desserialização direta da memória para um arquivo binário.

Justificativa: A serialização binária é o método mais eficiente para preservar o estado completo de estruturas complexas em C, especialmente aquelas que contêm ponteiros para dados alocados dinamicamente (como o mapa e o vetor de fantasmas). O formato binário garante uma cópia exata do estado da GameState, permitindo uma restauração precisa e rápida do jogo.

3. Divisão de responsabilidades e fluxo de trabalho detalhado

O projeto foi dividido em domínios de responsabilidade claros, permitindo o desenvolvimento paralelo e a integração eficiente dos componentes. A seguir, detalha-se o fluxo de trabalho e as entregas de cada integrante.

3.1. Contribuição de Gustavo: infraestrutura, estado, mapa e persistência

O trabalho do Gustavo concentrou-se na construção da infraestrutura de dados, no gerenciamento de memória e nos mecanismos de persistência do jogo.

Passo a passo da implementação:

I. Definição do estado central (GameState): Definiu a estrutura GameState (game.h), incluindo o mapa, Pac-Man, vetor dinâmico de fantasmas, pontuação, vidas, nível, timers e estado do menu. Implementou game_init() para inicializar o estado (score, vidas, nível, flags de pausa) e carregar o primeiro mapa.

Implementou game_load_level() para carregar um novo mapa, posicionar o Pac-Man, alocar dinamicamente o vetor de fantasmas com base nas posições encontradas no mapa e resetar contadores de pellets. Implementou game_shutdown() para liberar corretamente a memória dinâmica (mapa e vetor de fantasmas).

II. Gerenciamento dinâmico de mapas (map.h/.c): Definiu a estrutura Map para armazenar as dimensões (20x40), o vetor dinâmico de células (char*), posições iniciais de entidades e contagem de pellets. Implementou map_load() para ler arquivos de texto (mapa1.txt, mapa2.txt), alocar dinamicamente o mapa e os vetores de posições, e identificar as posições iniciais de Pac-Man ('P'), Fantasmas ('F'), Portais ('T'), Pellets ('.') e Power Pellets ('o'), calculando pelletsInitial e pelletsRemaining. Implementou map_free() para liberar toda a memória associada ao mapa (células, vetores de fantasmas e portais).

III. Controle de nível e HUD: Desenvolveu a lógica de avanço de nível em game.c, acionada quando pelletsRemaining atinge zero. Implementou as funções de desenho do HUD em render.c, exibindo vidas, pontuação, nível e a contagem de pellets restantes.

IV. Persistência de jogo (save.h/.c): Implementou as funções save_game() e load_game() para serialização e desserialização binária do estado completo do jogo. O arquivo savegame.sav armazena o mapa, pellets consumidos, posições/estados do Pac-Man e fantasmas, além da fase e placar, permitindo retomar a partida exatamente do ponto salvo.

3.2 Contribuição de Yasmyn: Interação, lógica de entidades e gráficos

O trabalho de Yasmyn focou na interação com o usuário, na lógica de movimento das entidades e na representação visual do jogo.

Passo a passo da implementação:

I. Loop principal e entrada (main.c e game.c): Configurou o loop principal em main.c, utilizando InitWindow() para criar a janela com a resolução baseada em MAP_COLS * TILE_SIZE e MAP_ROWS * TILE_SIZE + HUD_HEIGHT. Implementou a função game_update() em game.c (linhas 11-405), que agora contém o loop completo do jogo:

- Leitura de entrada (setas/WASD).
- Processamento do movimento discreto (4 blocos/s).
- Lógica de colisão e teleporte em portais.
- Coleta de pellets/power pellets e temporizador de power mode.
- Lógica de IA dos fantasmas e captura. Implementou o tratamento da tecla TAB para abrir/fechar o menu, pausando o jogo quando o menu está ativo.

II. Movimento e colisão do Pac-Man: Desenvolveu a lógica de movimento discreto do Pac-Man, controlada por moveTimer e pendingDir para garantir a taxa de 4 blocos/s e a mudança de direção no centro do bloco. Implementou a detecção de colisão com paredes e a funcionalidade de teletransporte através dos portais laterais.

III. Interação com pellets e pontuação: Implementou a lógica de consumo de pellets e power pellets, atualizando o score e decrementando pelletsRemaining. Adicionou a lógica de captura de fantasmas (+100 pontos) durante o modo vulnerável.

IV. Lógica dos fantasmas (IA e vulnerabilidade): Implementou o algoritmo de movimento simples dos fantasmas: seguir a direção até uma interseção e escolher uma nova direção válida de forma aleatória, com a restrição de não reverter imediatamente. Desenvolveu a lógica do modo vulnerável (timer de 8s, redução da velocidade, cor branca) e a remoção do fantasma ao ser comido.

V. Renderização e menu (render.h/.c e menu.h/.c): Implementou o módulo render.h/.c para desenhar o mapa (paredes azuis, pellets/power pellets, portais e piso escuro), o Pac-Man (círculo amarelo/dourado) e os fantasmas (círculos vermelhos ou brancos). Desenvolveu o módulo menu.h/.c para gerenciar o estado do menu, com navegação por setas/ENTER e atalhos N, C, S, Q, V. O menu é exibido como uma tela semitransparente com as opções destacando a seleção.