Projeto C — Relatório Técnico

1. Introdução

Este documento apresenta o relatório técnico do Projeto C, um jogo desenvolvido em linguagem C como parte de um projeto acadêmico. O objetivo deste relatório é documentar a arquitetura, decisões de design, implementação e resultados dos testes realizados.

---

2. Descrição do Projeto

Título do Jogo: [Nome do Jogo]

Objetivo: [Descrição breve do que o jogador faz e qual é o goal principal]

Público-alvo: [Ex: estudantes de programação, jogadores casuais, etc.]

Plataforma: Console (terminal/CLI)

Linguagem: C (C99 ou C11)

---

3. Requisitos

3.1 Requisitos Funcionais

- [RF1] O usuário pode [funcionalidade X]
- [RF2] O sistema deve [funcionalidade Y]
- [RF3] O jogo deve [funcionalidade Z]

3.2 Requisitos Não-Funcionais

- **Performance:** [Ex: resposta em < 100ms]
- **Usabilidade:** [Ex: interface intuitiva via menu de texto]
- **Portabilidade:** Compatível com Linux, Windows, macOS
- **Manutenibilidade:** Código modularizado em múltiplos arquivos `.c` e `.h`

---

4. Arquitetura do Sistema

4.1 Estrutura de Diretórios

```
Projeto-C/
├── src/
│   ├── main.c # Ponto de entrada do programa
│   ├── game.c # Lógica principal do jogo
│   ├── player.c # Funções relacionadas ao jogador
│   ├── [outro_modulo].c # [Descrição]
│   └── ...
├── include/
```

```
│   ├── game.h # Headers dos módulos
│   ├── player.h
│   ├── [outro_modulo].h
│   └── ...
├── Makefile # Build automation
├── README.md # Instruções de uso
└── [Este Relatório]
```

4.2 Módulos Principais

Módulo	Arquivo(s)	Responsabilidade
Main	main.c	Inicialização do programa e loop principal
Game	game.c / game.h	Lógica central do jogo, estado, turnos
Player	player.c / player.h	Estrutura e operações do jogador
[Módulo X]	[modulo_x].c / .h	[Descrição]

4.3 Estruturas de Dados Principais

Estrutura Player

```
typedef struct {
char name[50];
int health;
int mana;
int level;
// ... outros campos
} Player;
```

Estrutura Game

```
typedef struct {
Player player;
// ... estado do jogo
int turn;
// ... outros campos
} GameState;
```

---

5. Decisões de Design

5.1 Arquitetura

- **Modularização:** Código separado em módulos independentes (`.c` / `.h`) para facilitar manutenção e reutilização.
- **Paradigma:** Procedural com estruturas de dados (C não é orientado a objetos).
- **Gerenciamento de Memória:** [Estático / Dinâmico] — descrição de como aloca/libera memória.

5.2 Armazenamento de Estado

- Uso de estruturas globais ou passadas por referência entre funções.
- [Se houver persistência] Dados salvos em arquivo `.txt` / `.bin`.

5.3 Interface com Usuário

- Menu de navegação baseado em texto.
- Entrada via `scanf()` / `fgets()`.
- Output via `printf()`.

5.4 Algoritmos Principais

[Algoritmo 1: Nome]

- Complexidade: O(n)
- Descrição: [O que faz, como funciona]

[Algoritmo 2: Nome]

- Complexidade: O(n log n)
- Descrição: [O que faz, como funciona]

---

6. Fluxo Principal do Jogo

Início
↓
Inicializar estado do jogo
↓
Mostrar menu
↓
[Loop de Jogo]
├— Processar entrada do usuário
├— Atualizar estado
├— Renderizar tela
└— Continuar se não terminou
↓
Fim do jogo (vitória / derrota)
↓
Exibir resultado
↓
Salvar dados (opcional)

↓
Fim

---

7. Implementação

7.1 Principais Funções

Função	Assinatura	Descrição
`init_game()`	`void init_game(GameState *state)`	Inicializa o estado do jogo
`update_game()`	`void update_game(GameState *state, int input)`	Atualiza lógica a cada turno
`render_game()`	`void render_game(GameState *state)`	Exibe o estado atual na tela
`player_attack()`	`int player_attack(Player *p)`	Calcula dano do ataque

7.2 Tratamento de Erros

- Validação de entrada do usuário com `if` e `while` loops.
- Checagem de alocação dinâmica com `if (ptr == NULL)`.
- Mensagens de erro descritivas ao usuário.

7.3 Bibliotecas Utilizadas

- `<stdio.h>` — Input/Output
- `<stdlib.h>` — Alocação de memória
- `<string.h>` — Manipulação de strings
- `<time.h>` — Geração de números aleatórios (se aplicável)
- [Outras bibliotecas usadas]

---

8. Testes

8.1 Testes Unitários

- [Teste 1] Verificar se `init_game()` inicializa corretamente.
- [Teste 2] Verificar se `player_attack()` calcula dano dentro do intervalo esperado.
- [Teste 3] Validar entrada inválida do usuário.

8.2 Testes de Integração

- Teste do fluxo completo de uma partida.
- Teste de transições entre estados do jogo.
- Teste de menu de navegação.

8.3 Testes de Aceitação

- Jogo deve rodar sem crashes por 10+ minutos.
- Vitória/derrota devem ser alcançáveis.
- Todas as funcionalidades mencionadas nos requisitos funcionam.

---

9. Limitações e Melhorias Futuras

9.1 Limitações Atuais

- Interface apenas em texto (sem gráficos).
- Sem persistência de dados (não salva progresso).
- Sem multiplayer/rede.
- Sem trilha sonora ou efeitos sonoros.

9.2 Melhorias Propostas

1. **Gráficos:** Integrar biblioteca como `ncurses` para melhor visualização.
2. **Persistência:** Implementar save/load de progresso em arquivo.
3. **Múltiplos Personagens:** Permitir criação e seleção de diferentes personagens.
4. **IA Inimiga:** Implementar lógica de IA para adversários.
5. **Leaderboard:** Sistema de pontuação e ranking.

---

10. Conclusão

O Projeto C demonstra aplicação prática de conceitos fundamentais de programação em C, incluindo estruturas de dados, modularização, manipulação de entrada/saída e lógica de programação. O jogo é funcional e atende aos requisitos propostos, oferecendo base sólida para futuras expansões e melhorias.