# Relatório Técnico do Projeto

## Jogo da Vida de Conway em Linguagem C

**Aluno:** Gabriel Ferreira Cavalcante
**Matrícula:** 20250035736
**Disciplina:** Introdução às Técnicas de Programação (ITP)
**Unidade:** 1
**Data de entrega:** 30/09/2025 (terça-feira)

### 1. Resumo

Este relatório detalha o processo de desenvolvimento e a estrutura técnica do projeto final da disciplina de Introdução à Tecnologia da Programação, que consiste em uma implementação do "Jogo da Vida" de John Conway. O projeto foi desenvolvido inteiramente na linguagem C, com compilação via GCC, e opera através de uma interface de linha de comando (CLI).

O Jogo da Vida é um autômato celular que simula a evolução de uma colônia de células em um tabuleiro bidimensional. O estado de cada célula (viva ou morta) em uma geração é determinado por um conjunto de quatro regras simples baseadas na quantidade de células vizinhas vivas. Apesar da simplicidade de suas regras, o sistema exibe comportamentos emergentes complexos, criando padrões estáticos, oscilatórios e móveis.

A implementação desenvolvida não apenas simula a evolução das gerações, mas também oferece funcionalidades interativas para o usuário. O programa inclui um menu inicial que permite a escolha entre padrões clássicos pré-definidos (como "Glider" e "Blinker") ou a entrada em um "Modo de Edição Manual", onde o usuário pode definir o estado inicial do tabuleiro, adicionando células vivas em coordenadas específicas. Este recurso aumenta a rejogabilidade e permite a exploração de diferentes configurações.

O objetivo principal do projeto foi aplicar e solidificar os conceitos fundamentais de programação abordados na primeira unidade da disciplina, incluindo a manipulação de variáveis, o uso de operadores, a implementação de estruturas condicionais e de repetição, a modularização do código com funções e a utilização de vetores (matrizes) como estrutura de dados central. O resultado é um programa funcional, robusto e bem-estruturado que cumpre todos os requisitos propostos.

## 2. Análise Técnica

### 2.1. Metodologias e Tecnologias

O desenvolvimento do projeto seguiu uma abordagem de design modular e incremental. A complexidade do problema foi decomposta em partes menores e mais gerenciáveis, refletidas na estrutura de funções do código. O processo foi guiado pelas sprints propostas no plano de disciplina, onde cada etapa adicionava uma nova camada de funcionalidade sobre a anterior:

1. **Sprint 1:** Foco na estrutura de dados (tabuleiro) e na saída básica (visualização inicial).
2. **Sprint 2:** Implementação da lógica central do jogo (regras de evolução) e da interação mínima.
3. **Sprint 3:** Refatoração do código em funções especializadas, garantindo clareza e manutenibilidade.

A "pilha tecnológica" (*stack*) utilizada foi minimalista e focada nos fundamentos da computação, consistindo em:

- **Linguagem de Programação: C**. A escolha da linguagem C foi um requisito da disciplina e se mostrou ideal para o projeto. Por ser uma linguagem de nível mais baixo, ela forçou uma atenção deliberada à gestão de memória (através de arrays estáticos) e à manipulação de tipos de dados. A performance do C permitiu que as simulações, mesmo em tabuleiros maiores, fossem computadas de forma praticamente instantânea.
- **Compilador: GCC (GNU Compiler Collection)**. Utilizado para compilar o código-fonte (`main.c`) em um programa executável. O comando de compilação utilizado foi `gcc src/main.c -o JogoDaVida`, demonstrando o processo padrão de compilação em um ambiente de desenvolvimento C.
- **Ambiente de Execução: Interface de Linha de Comando (CLI)**. Toda a interação do programa ocorre no terminal. A visualização do tabuleiro é feita com caracteres (`#` para células vivas, `.` para mortas), e a entrada de dados do usuário é capturada pelas funções `scanf` e `getchar`.

### 2.2. Arquitetura do Software

A solução foi arquitetada em torno de uma estrutura de dados central: uma matriz bidimensional de inteiros. Um dos principais desafios técnicos do Jogo da Vida é a necessidade de uma atualização "simultânea" de todas as células, ou seja, o cálculo do estado de uma célula na próxima geração deve ser baseado inteiramente no estado da geração atual, sem ser influenciado por células que já foram atualizadas no mesmo ciclo.

Para resolver este problema, a arquitetura emprega dois tabuleiros (matrizes) idênticos:

1. `tabuleiro`: Armazena o estado da geração atual, servindo apenas para leitura.
2. `proximoTabuleiro`: Um buffer que armazena o estado calculado da próxima geração.

O fluxo de execução principal, orquestrado pela função `main`, segue um ciclo claro a cada geração:

1. **Imprimir:** O estado do `tabuleiro` atual é exibido na tela.
2. **Calcular:** O programa percorre o `tabuleiro` atual, aplica as regras e escreve os novos estados no `proximoTabuleiro`.
3. **Copiar:** O conteúdo do `proximoTabuleiro` é copiado de volta para o `tabuleiro` principal.
4. **Pausar:** O programa aguarda a interação do usuário para iniciar o próximo ciclo.

Essa abordagem garante a integridade da simulação e é uma solução clássica e eficiente para este tipo de problema.

### 2.3. Aplicação dos Conceitos da Unidade 1

O projeto foi cuidadosamente estruturado para aplicar todos os conceitos de programação exigidos.

- **Variáveis e Tipos de Dados:** Foram utilizadas variáveis de tipos `int` (para contadores, coordenadas e para o estado do tabuleiro), `char` (para a representação visual das células) e `char[]` (para armazenar o título de cada geração). O uso de `#define` para constantes como `ALTURA`, `LARGURA` e os caracteres das células demonstra a aplicação de boas práticas para evitar "números mágicos" no código.
- **Estruturas Condicionais (`if/else`, `switch`):**
    - O `if/else` é a base da lógica do jogo. É utilizado na função `calcularProximaGeracao` para implementar as quatro regras de sobrevivência, morte e nascimento.
    - É também usado em `contarVizinhosVivos` para garantir que as coordenadas dos vizinhos não extrapolem os limites da matriz, tratando os casos de borda.
    - O comando `switch` foi implementado no menu inicial (`inicializarTabuleiro`), permitindo ao usuário escolher entre diferentes padrões ou o modo de edição, demonstrando o uso de seletores de múltipla escolha.

```
// Exemplo de uso de if/else nas regras do jogo
if (estadoAtual == 1 && (vizinhosVivos == 2 || vizinhosVivos == 3)) {
    proximoTabuleiro[i][j] = 1; // Sobrevivência
}
if (estadoAtual == 0 && vizinhosVivos == 3) {
```

```
    proximoTabuleiro[i][j] = 1; // Nascimento
}
```

- **Estruturas de Repetição (`for`, `while`):**
    - Laços `for` aninhados são a estrutura mais utilizada no projeto, sendo essenciais para percorrer a matriz bidimensional em praticamente todas as funções: `inicializarTabuleiro`, `imprimirTabuleiro`, `contarVizinhosVivos`, `calcularProximaGeracao` e `copiarTabuleiro`.
    - Um laço `while (1)` foi utilizado para criar o loop do "Modo de Edição Manual", que só é interrompido quando o usuário digita `-1`.
    - Outro `while` foi usado na função `esperarEnter` para limpar o buffer de entrada (`stdin`), resolvendo um problema comum de entrada de dados em C.

```
// Exemplo de uso de for aninhado
for (int i = 0; i < ALTURA; i++) {
    for (int j = 0; j < LARGURA; j++) {
        // ... lógica para a célula (i, j) ...
    }
}
```

- **Vetores (Arrays):**
    - A estrutura de dados central é um vetor bidimensional (`int tabuleiro[ALTURA][LARGURA]`), que representa o mundo do jogo. Sua aplicação é um exemplo claro de como matrizes são ideais para representar dados espaciais. O uso de dois vetores (`tabuleiro` e `proximoTabuleiro`) para gerenciar o estado demonstra uma compreensão mais profunda do problema algorítmico.
- **Funções:**
    - O código foi extensivamente modularizado em **6 funções** além da `main`. Cada função possui uma responsabilidade única e bem-definida, o que torna o código mais legível, organizado e fácil de depurar.
        - `inicializarTabuleiro`: Gerencia toda a configuração inicial.
        - `imprimirTabuleiro`: Encapsula a lógica de visualização.
        - `contarVizinhosVivos`: Isola o algoritmo de contagem de vizinhos.
        - `calcularProximaGeracao`: Contém a lógica principal das regras do jogo.
        - `copiarTabuleiro`: Realiza a tarefa repetitiva de atualizar o tabuleiro.
        - `esperarEnter`: Abstrai a complexidade da pausa e limpeza do buffer.

## 3. Conclusão e Reflexão

A implementação do projeto "Jogo da Vida de Conway" foi uma experiência de aprendizado extremamente valiosa e gratificante. Vindo de um contexto de programação mais voltado para a web (do curso técnico em Informática para Internet), mergulhar na linguagem C para este projeto me proporcionou uma compreensão muito mais profunda e fundamental da computação.

O principal desafio técnico encontrado foi, sem dúvida, a necessidade de garantir a atualização síncrona do tabuleiro. A percepção de que modificar o tabuleiro principal enquanto se calcula a próxima geração corromperia a simulação levou à solução de utilizar uma matriz auxiliar. Este problema ilustrou na prática a importância de gerenciar o "estado" de um sistema de forma cuidadosa e deliberada. Outro desafio menor, mas igualmente educativo, foi lidar com o buffer de entrada do terminal, o que exigiu a criação de uma função `esperarEnter` robusta.

O maior aprendizado foi a constatação do poder da modularização. Decompor um problema aparentemente complexo em funções pequenas e focadas não apenas tornou o código mais organizado, mas também simplificou drasticamente o processo de desenvolvimento e depuração. Cada função pôde ser testada de forma isolada, garantindo que a base do sistema estivesse sólida antes de integrar as partes.

Este projeto serviu como uma ponte perfeita entre a teoria da sala de aula e a prática da engenharia de software. Conceitos como laços, vetores e condicionais deixaram de ser apenas comandos de sintaxe e se tornaram ferramentas para construir um sistema dinâmico e funcional. A transição para o BTI, começando com um projeto tão fundamental em C, solidificou as bases necessárias para os desafios mais avançados que virão no curso. O resultado final é um programa do qual me orgulho, não apenas por sua funcionalidade, mas pelo conhecimento consolidado durante sua criação.