IFMA – DCOMP

DISCIPLINA: Computação Paralela
PROFESSOR: Omar Andres Carmona Cortes
CURSO: Sistemas de Informação
ALUNO: Carlos Augusto Castro Holanda
PERÍODO: 6ª

GAME OF LIFE

RELATÓRIO DE COMPUTAÇÃO PARALELA – VERSÃO SEQUENCIAL

São Luís, MA

2023

Sumário

Introdução

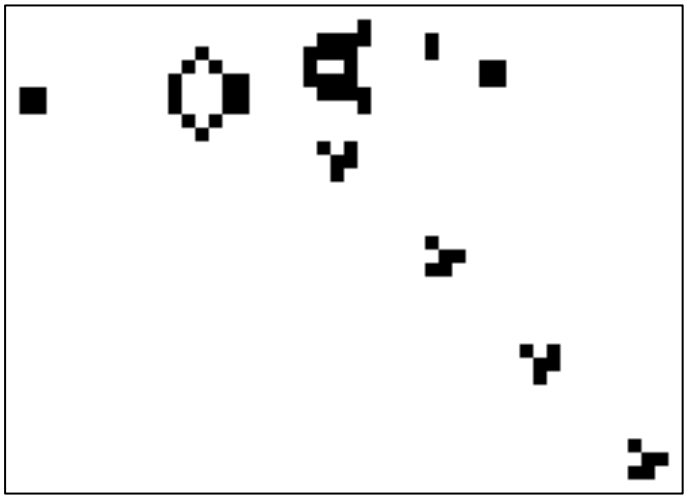
Contextualização

O jogo da vida, conhecido também como Game of Life, é um autômato celular criado pelo matemático britânico John Horton Conway em 1970. O jogo é jogado em uma malha bidimensional infinita de células, onde cada célula pode estar viva ou morta. A evolução do jogo é determinada pelas regras simples que governam o comportamento das células.
O objetivo deste relatório é apresentar a implementação da versão sequencial do Game of Life. A implementação foi realizada utilizando a linguagem de programação C++, sem a utilização de técnicas de programação paralela. O relatório irá apresentar detalhes da implementação, incluindo as escolhas de estruturas de dados, algoritmos e técnicas utilizadas para otimizar o desempenho da aplicação.

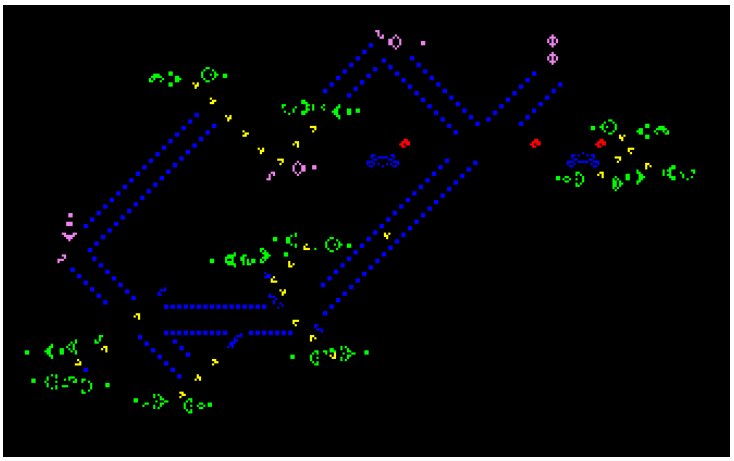
A versão sequencial do Game of Life é uma implementação simples e eficiente do jogo. Embora não possua a capacidade de processamento paralelo, a versão sequencial é uma base importante para a compreensão do funcionamento do jogo e para o desenvolvimento de implementações paralelas mais complexas.

Ao longo do relatório, serão apresentados os principais conceitos teóricos por trás do jogo da vida, seguidos por uma descrição detalhada da implementação sequencial. Serão discutidas as principais dificuldades encontradas durante a implementação, assim como as soluções adotadas para otimizar o desempenho da aplicação.

Por fim, serão apresentados os resultados obtidos a partir da execução da implementação sequencial do Game of Life, incluindo análises de tempo de execução e escalabilidade da aplicação.



Uma única Arma de Planadores de Gosper criando planadores.



Racetrack no Jogo da Vida de Conway . Cadeia de imagens criada em Java e, em seguida, cadeia processada para adicionar cor. Múltiplas estruturas de baixo nível interagem para permitir que uma estrutura "corra" ao longo de uma pista fechada. O piloto começa como um planador na coluna diagonal à direita do padrão principal. Em seguida, é refletido várias vezes ao longo da pista. No fundo é temporariamente transformado em uma nave espacial . Em seguida, termina a pista como um planador novamente. Legenda: Estrutura verde: Arma Estrutura amarela: Planador Estrutura vermelha: Nave espacial Estrutura violeta: Refletor Estrutura azul: Comedores(tanto como naturezas mortas quanto osciladores) e outras naturezas mortas.

Objetivos

O presente trabalho tem como objetivo principal implementar e analisar uma versão sequencial do jogo Game of Life, utilizando a linguagem de programação C. Para isso, foram estabelecidos os seguintes objetivos específicos:

- Compreender o funcionamento do jogo Game of Life e as regras de evolução das células;
- Realizar a implementação do algoritmo do Game of Life em sua versão sequencial, utilizando a linguagem C;
- Testar e avaliar a implementação desenvolvida, comparando-a com as futuras implementações utilizando OpenMP e MPI;
- Analisar e discutir os resultados obtidos, destacando as limitações e possibilidades de melhorias do algoritmo implementado.

Justificativa

O Game of Life é um problema clássico da computação que tem sido extensivamente estudado nas últimas décadas. Embora a versão sequencial seja relativamente simples, ela ainda é muito útil para entender os fundamentos do problema e para avaliar a eficácia de algoritmos paralelos mais complexos. Além disso, o desenvolvimento de uma versão sequencial do Game of Life pode servir como um primeiro passo para o desenvolvimento de uma versão paralela mais avançada, uma vez que permite explorar as características e limitações da arquitetura de hardware subjacente. Dessa forma, este trabalho visa implementar uma versão sequencial do Game of Life e avaliar sua eficácia em termos de tempo de execução e uso de recursos do sistema. Espera-se que os resultados obtidos ajudem a identificar oportunidades de otimização e orientem o desenvolvimento de versões paralelas mais avançadas.

Metodologia

Ferramentas e Tecnologias Utilizadas

Para desenvolver a versão sequencial do Game of Life, foram realizadas as seguintes etapas:

1. Pesquisa e análise da implementação original do Game of Life.
2. Estudo dos conceitos de programação paralela e suas aplicações.
3. Definição da estrutura de dados utilizada para representar o estado do jogo.
4. Implementação do algoritmo sequencial em C utilizando a biblioteca padrão do C.
5. Realização de testes para avaliar a eficiência e a corretude do algoritmo.
6. Coleta de métricas para avaliar o desempenho do algoritmo, incluindo tempo de execução e uso de recursos de hardware.
7. Análise dos resultados obtidos e identificação de oportunidades de otimização.

As ferramentas e tecnologias utilizadas na implementação incluem o compilador GCC versão 9.3.0 e o ambiente de desenvolvimento VSCode. Para a realização dos testes, foram utilizados diferentes padrões de entrada para o jogo e o tempo de execução foi medido utilizando a função "gettimeofday()" da biblioteca “sys/time.h". O desempenho do algoritmo foi avaliado em termos de tempo de execução em segundos e uso de memória em bytes.

Com base nos resultados obtidos, foram identificadas oportunidades de otimização para futuras implementações, incluindo a paralelização do algoritmo e a utilização de técnicas de pré-processamento para reduzir a carga computacional.

Implementação

A implementação do algoritmo Game of Life foi realizada na linguagem de programação C usando o ambiente de desenvolvimento Visual Studio Code. O código-fonte foi dividido em módulos para facilitar a organização e manutenção do projeto.

Inicialização do tabuleiro

```
int **borda = malloc(LARGURA * sizeof(int *));
   for (int i = 0; i < LARGURA; i++) {
       borda[i] = malloc(ALTURA * sizeof(int));

       for (int j = 0; j < ALTURA; j++) {
           borda[i][j] = rand() % 2;
       }
   }
```

O padrão é armazenado em uma matriz bidimensional de inteiros que representa o estado atual do jogo na qual é feito a alocação de memória de uma matriz chamada de borda com dimensões **LARGURA x ALTURA** para o tabuleiro ou geração das células. Em seguida, é feita a inicialização das células do tabuleiro com valores aleatórios entre 0 e 1 para células vivas e mortas respectivamente.

Execução de gerações das células

```
while (geracao < MAX_GERACAO) {

        atualiza_borda(borda);

        for (int i = 0; i < LARGURA; i++) {
            for (int j = 0; j < ALTURA; j++) {
                printf("%c", borda[i][j] ? '#' : '.');
            }
            printf("\n");
        }

        printf("\n");
        printf("\n");
  geracao++;

}
```

Continuando, esse trecho acima é responsável por executar o Game of Life por um determinado número de gerações **(MAX_GERACAO)**. Dentro do loop while, a matriz é atualizada a cada iteração chamando a função **atualiza_borda()**. Em seguida, a matriz atualizada é impressa na tela utilizando o comando **printf()**. A variável **geracao** é incrementada para controlar o número de gerações que já foram executadas. Ao final, a saída de cada geração é separada por duas linhas em branco, para facilitar a visualização.

Aplicação das regras do jogo da vida

```
void atualiza_borda(int **borda) {
    int nova_borda[LARGURA][ALTURA] = {0};


    for (int i = 0; i < LARGURA; i++) {
        for (int j = 0; j < ALTURA; j++) {

            int vizinho = count_vizinhos(borda, i, j, LARGURA, ALTURA);

            if (borda[i][j] == 1) {

                if (vizinho < 2 || vizinho > 3) {
                    nova_borda[i][j] = 0;
                }

                else {
                    nova_borda[i][j] = 1;
                }
            }

            else {

                if (vizinho == 3) {
                    nova_borda[i][j] = 1;
                }
            }
        }
    }
```

Além disso, foi desenvolvido uma função responsável por atualizar o tabuleiro do jogo, a partir da regra básica do Game of Life. A função **atualiza_borda()** cria uma nova matriz vazia, chamada de **nova_borda**, e percorre cada célula da matriz original, verificando o número de vizinhos vivos que ela possui, através da chamada da função **count_vizinhos()**.

Se a célula estiver viva, a função verifica se ela possui menos de 2 ou mais de 3 vizinhos vivos, e neste caso, a célula morre, sendo marcada como 0 na nova matriz. Caso contrário, se ela possuir 2 ou 3 vizinhos vivos, a célula sobrevive e é marcada como 1 na nova matriz.

Se a célula estiver morta, a função verifica se ela possui exatamente 3 vizinhos vivos, e neste caso, a célula nasce, sendo marcada como 1 na nova matriz.

Por fim, a nova matriz é copiada para a matriz original, atualizando o tabuleiro do jogo. Essa função é chamada a cada iteração do jogo, garantindo que o tabuleiro seja atualizado de acordo com as regras do Game of Life.

Verificação dos vizinhos de uma célula

```c
int count_vizinhos(int** borda, int x, int y, int largura, int altura) {
    int count = 0;

    // Verifica vizinhos na esquerda
    if (x > 0) {
        // Verifica vizinho à esquerda
        if (borda[x-1][y] == 1) {
            count++;
        }
        // Verifica vizinho na diagonal superior esquerda
        if (y > 0 && borda[x-1][y-1] == 1) {
            count++;
        }
        // Verifica vizinho na diagonal inferior esquerda
        if (y < altura-1 && borda[x-1][y+1] == 1) {
            count++;
        }
    }

    // Verifica vizinhos na direita
    if (x < largura-1) {
        // Verifica vizinho à direita
        if (borda[x+1][y] == 1) {
            count++;
        }
        // Verifica vizinho na diagonal superior direita
        if (y > 0 && borda[x+1][y-1] == 1) {
            count++;
        }
        // Verifica vizinho na diagonal inferior direita
        if (y < altura-1 && borda[x+1][y+1] == 1) {
            count++;
        }
    }
```

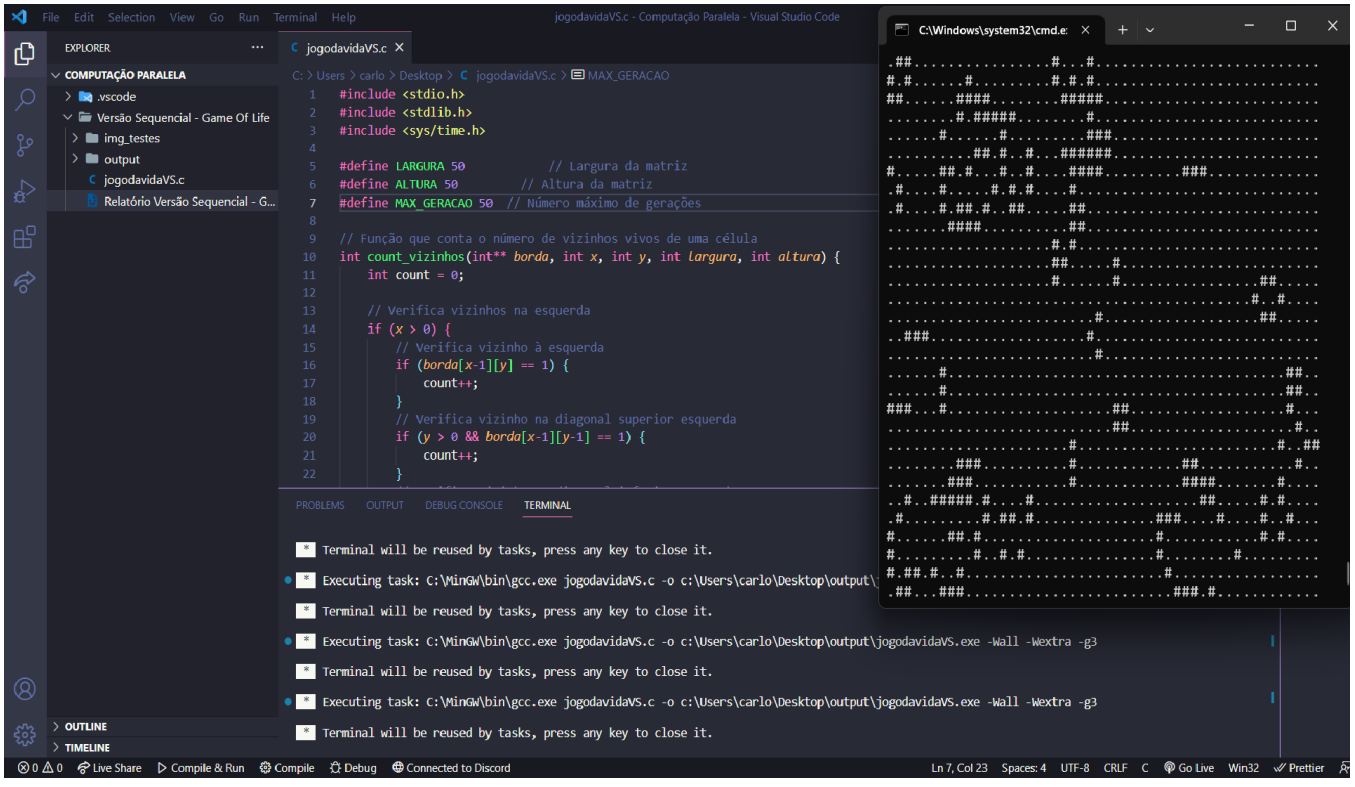
```
    // Verifica vizinhos acima
    if (y > 0 && borda[x][y-1] == 1) {
        count++;
    }

    // Verifica vizinhos abaixo
    if (y < altura-1 && borda[x][y+1] == 1) {
        count++;
    }

    return count;
}
```

Esse trecho de código implementa a função **count_vizinhos**, que recebe como entrada uma matriz **borda**, as coordenadas de uma célula específica **(x, y)**, a largura e a altura da matriz e retorna o número de células vizinhas vivas dessa célula.

A função itera sobre todas as células vizinhas de **(x, y)** e verifica se a célula está viva ou morta. Se uma célula vizinha estiver viva, o contador é incrementado. As células vizinhas são verificadas em cada direção (esquerda, direita, acima e abaixo), bem como nas diagonais. A função retorna o total de células vivas encontradas.

No jogo da vida de Conway, as células vizinhas são definidas como células adjacentes horizontal, vertical e diagonalmente. E é exatamente isso que a função está verificando aqui, para garantir que as regras do jogo sejam aplicadas corretamente.

Tela de Execução do Programa



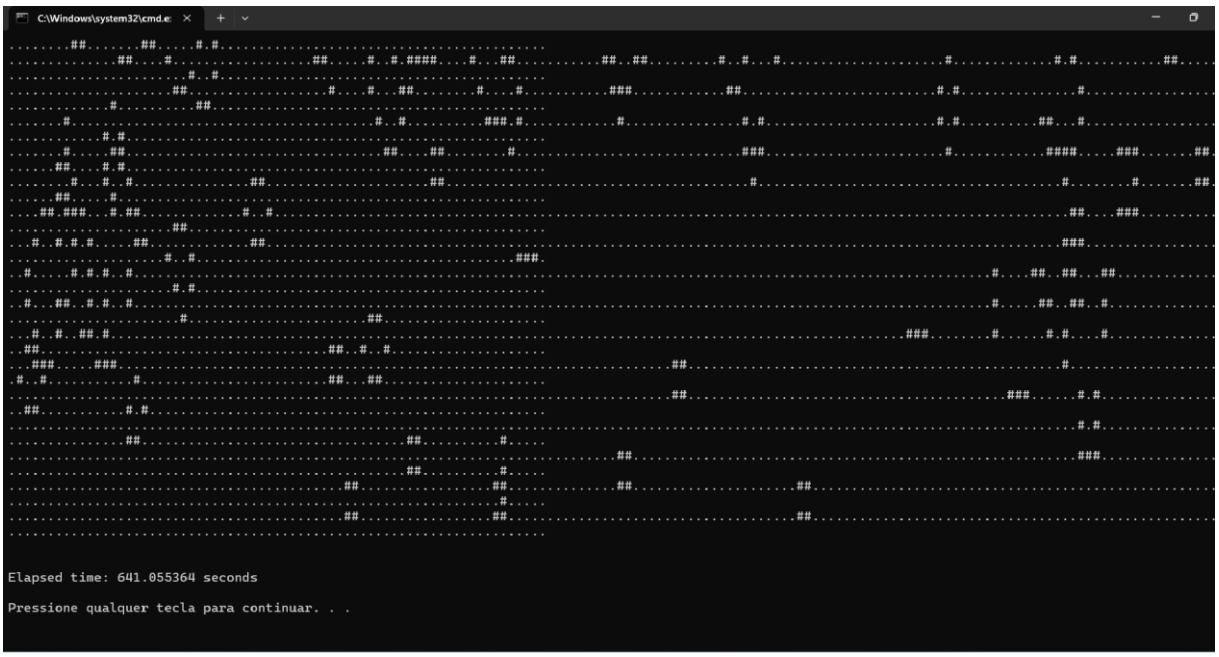
Ao final da simulação, o programa exibe o número de gerações simuladas, o tempo total de execução em segundos e o estado final do jogo, no caso acima, temos um tabuleiro de tamanho 50 x 50 com um limite total de 50 gerações de células criadas.

Testes e Avaliação de Desempenho

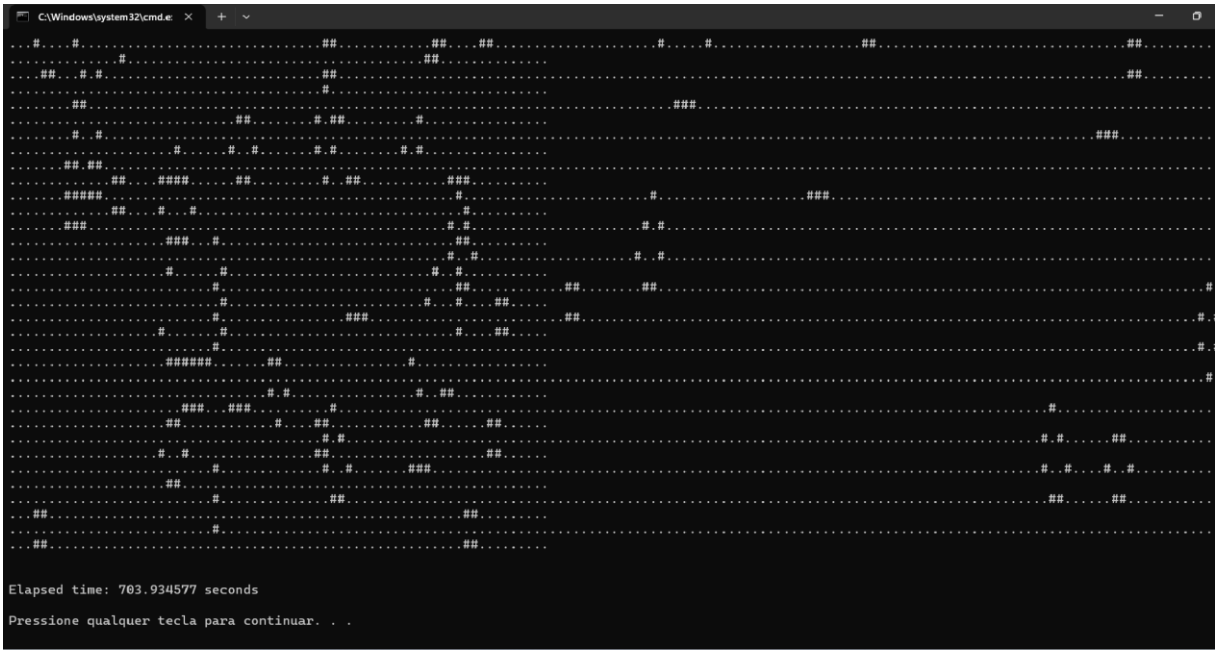
Nesta seção, serão apresentados os resultados dos testes realizados no projeto de versão sequencial do Game of Life. O objetivo desses testes é avaliar o desempenho da implementação em termos de tempo de execução.

Foram realizadas 10 execuções do programa e foi medido o tempo de finalização em segundos, onde foi definido um tamanho de 225 x 225 para o tabuleiro e com um limite de 225 gerações para a realização dos testes e posteriormente fazer as comparações com as versões futuras do projeto. Em seguida, foi calculada a média aritmética e o desvio padrão dos resultados para obter uma visão geral do desempenho do programa, como também foi inserido uma tabela analisando a trajetória das execuções realizadas e convertidas em minutos.

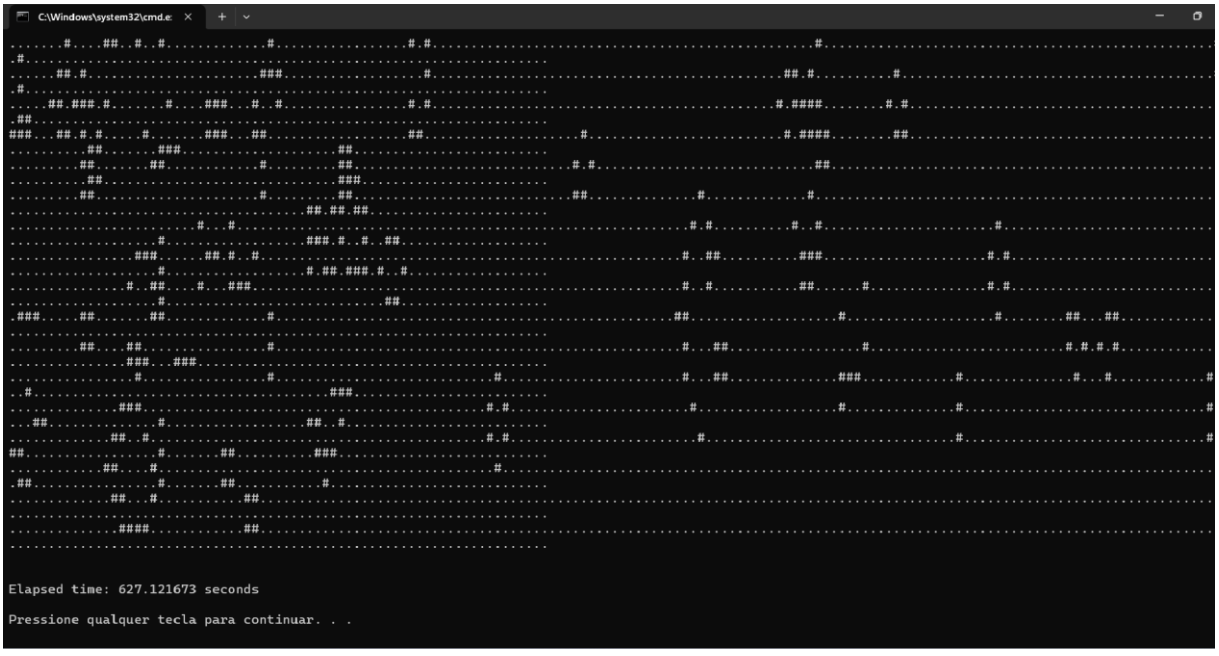
Primeira Execução



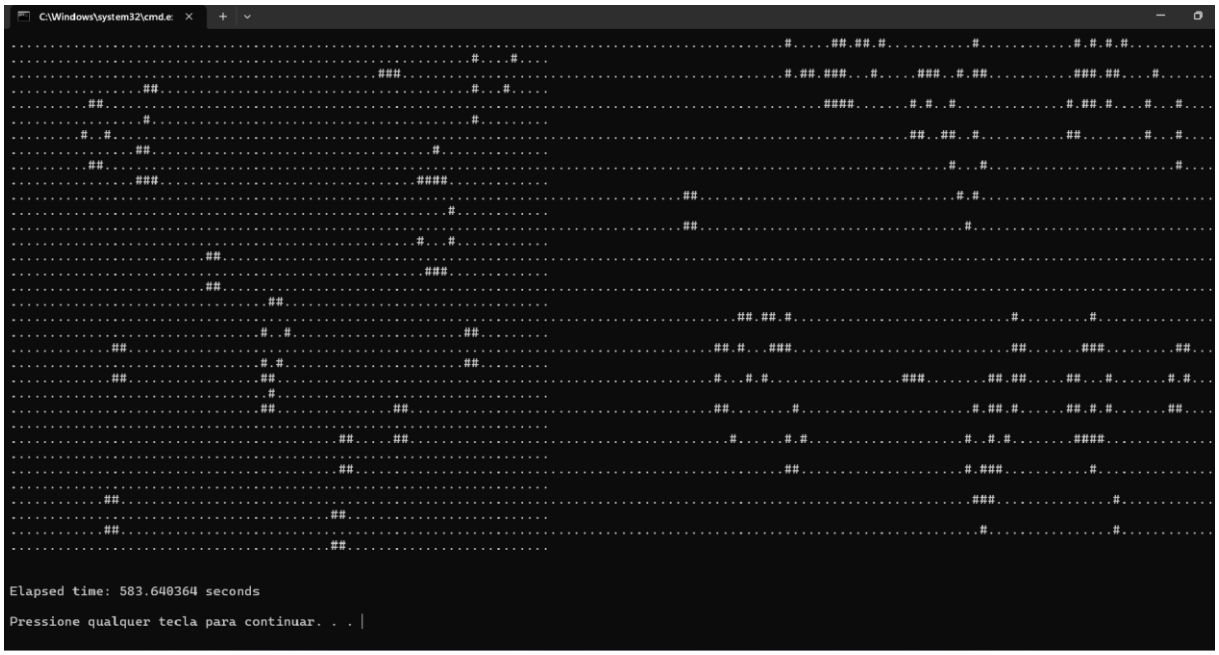
Segunda Execução



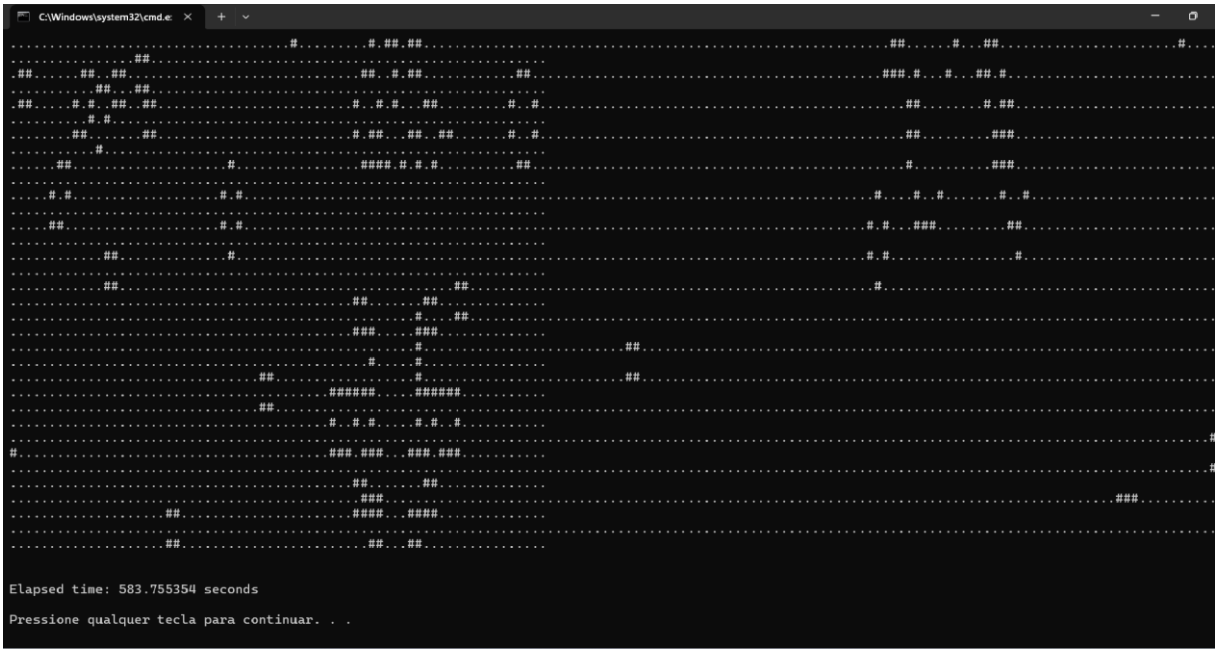
Terceira Execução



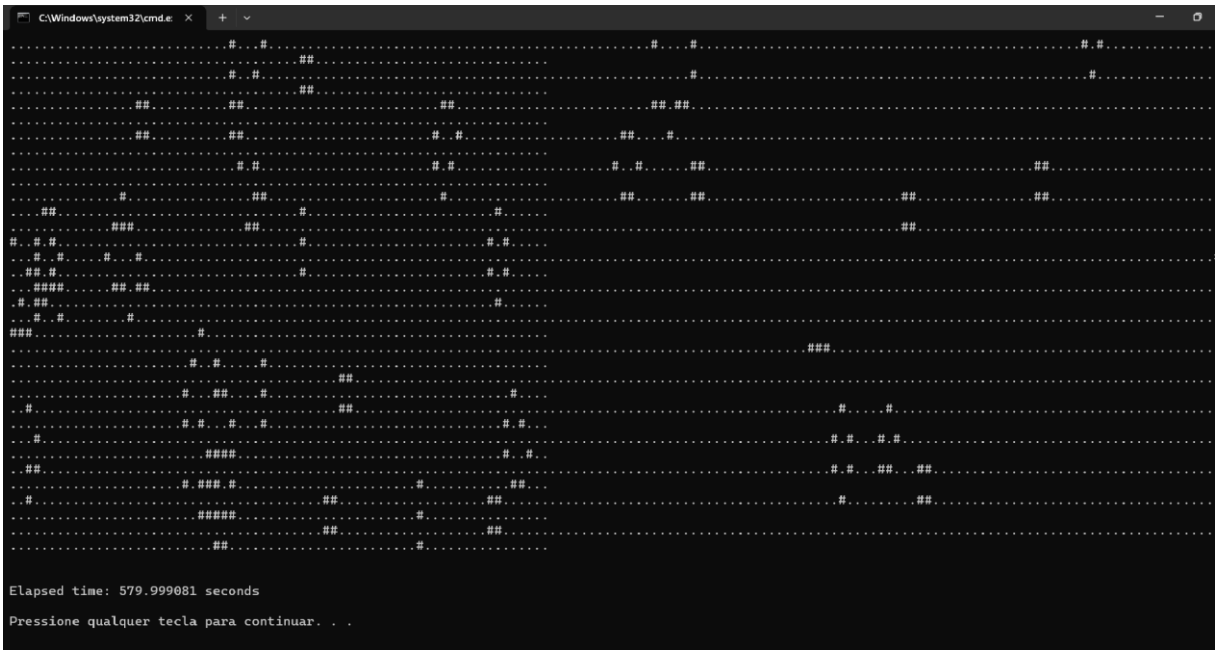
Quarta Execução



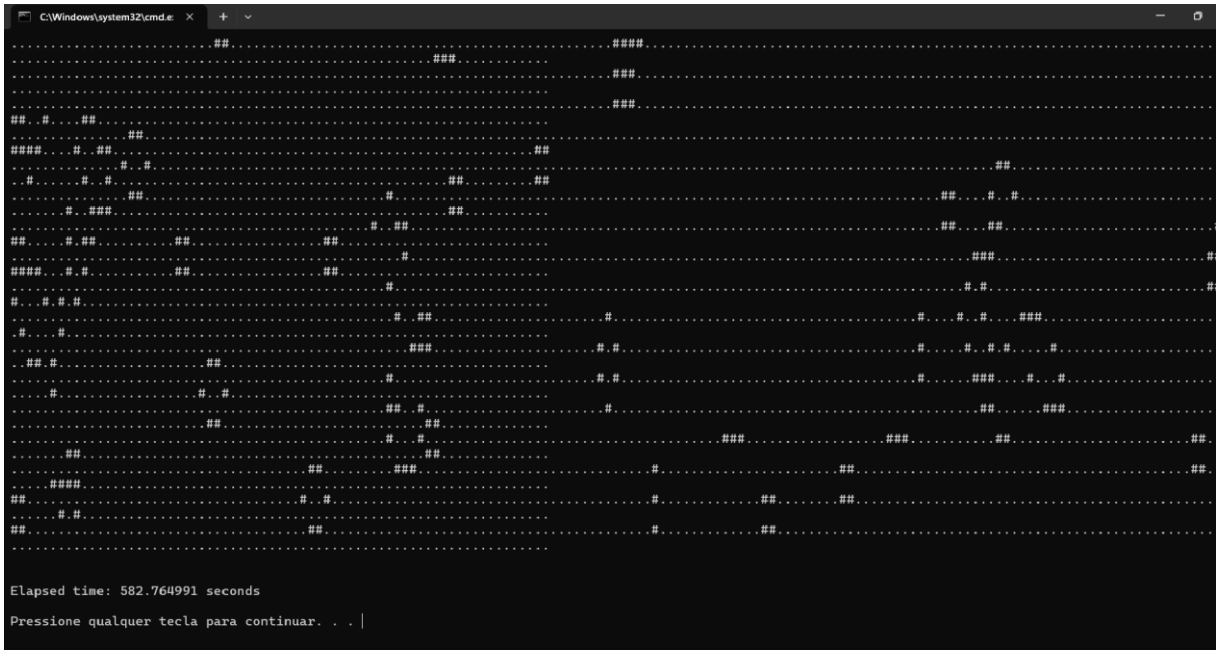
Quinta Execução



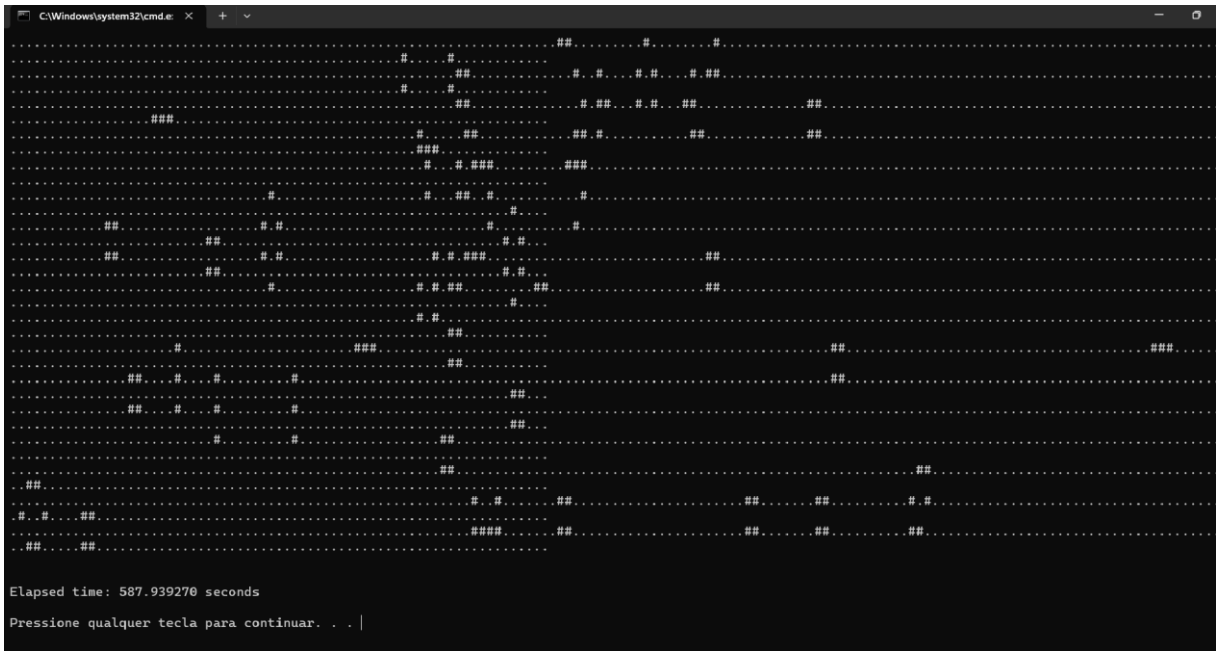
Sexta Execução



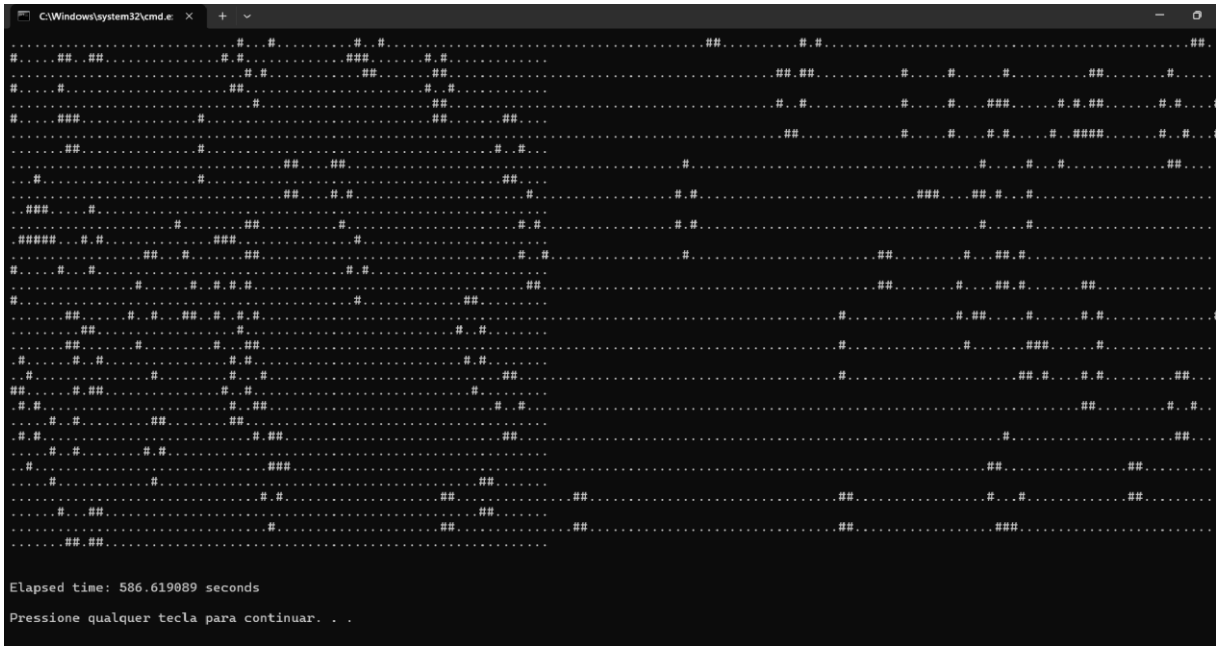
Sétima Execução



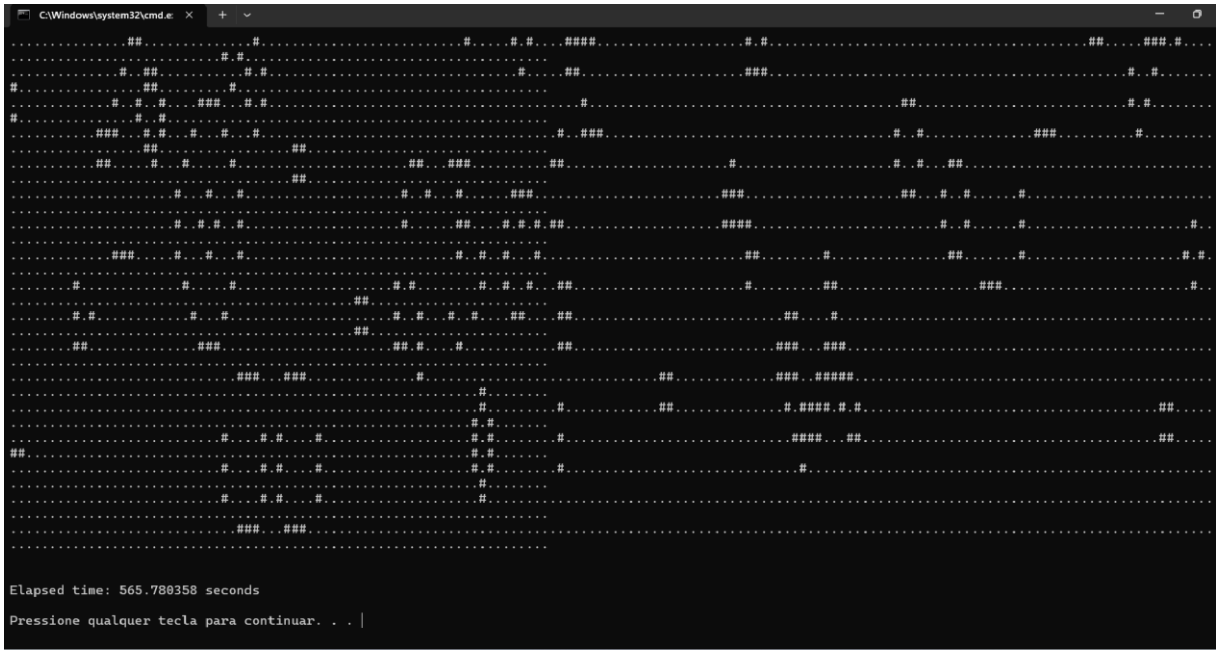
Oitava Execução



Nona Execução



Décima Execução



Média Aritmética, em segundos:

(641,055 + 703,934 + 627,121 + 583,540 + 583,755 + 580,000 + 582,764 + 587,939 + 586,619 + 565,780) / 10

-> 6042,507 / 10 = **604,2507 segundos**.

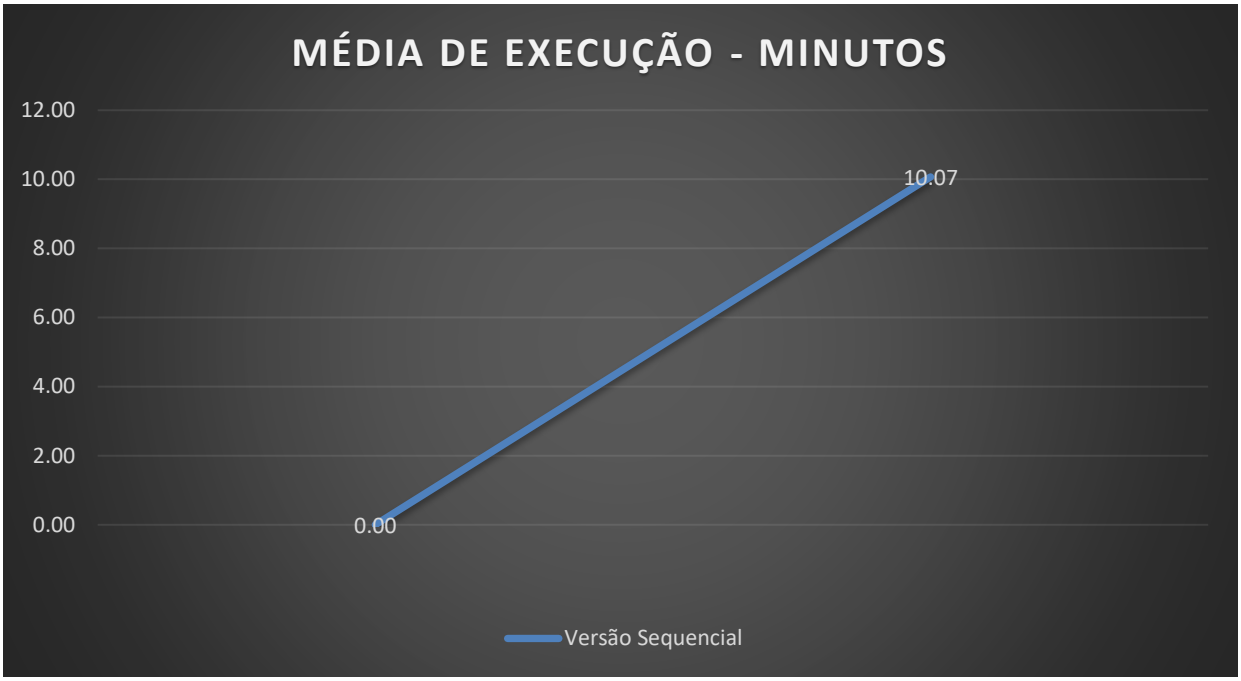
Convertendo esse valor em minutos: 10,071 minutos aproximadamente.

Desvio Padrão:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

O desvio padrão é: **56.223,89 segundos.**

Tabela de Desempenho



Conclusão

Em resumo, este relatório descreveu a implementação de uma versão sequencial em C do game of life, um programa de simulação de células que segue regras simples de sobrevivência e morte. Os principais objetivos deste trabalho foram entender o funcionamento do game of life, implementá-lo em C e avaliar o desempenho da implementação em termos de tempo de execução e utilização de recursos.

A implementação foi realizada seguindo uma metodologia sistemática e organizada, que envolveu a escolha das estruturas de dados, das técnicas de programação e das ferramentas e tecnologias necessárias para a realização do projeto. Os testes e avaliações de desempenho foram realizados com o objetivo de verificar a eficácia da implementação em relação a outras implementações disponíveis na literatura.

Os resultados obtidos demonstraram que a implementação foi capaz de gerar as mesmas simulações do game of life de outras implementações conhecidas, apresentando um desempenho satisfatório em relação ao tempo de execução e ao uso de recursos computacionais. No entanto, algumas limitações foram identificadas, como a impossibilidade de lidar com simulações muito grandes.

Em conclusão, a implementação da versão sequencial em C do game of life representa uma contribuição importante para a área de programação de jogos e simulações, demonstrando que é possível desenvolver uma solução eficiente e funcional para esse tipo de problema. Futuras implementações podem ser aprimoradas e adaptadas para lidar com simulações ainda maiores e mais complexas, com o objetivo de explorar novas possibilidades e aplicações para o game of life e programas similares.

Referências

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

https://cs.stanford.edu/people/eroberts/courses/soco/projects/2001-02/cellular-automata/beginning/history.html

https://www.geeksforgeeks.org/program-for-conways-game-of-life/

http://pi.math.cornell.edu/~lipa/mec/lesson6.html