**Dayman Moreira Novaes 29/05/2015**

Ciência da Computação 1º Período

Algoritmo e Estrutura de Dados Trabalho Prático

**JOGO DA VIDA**

**Introdução**

Jogo da Vida é um simples jogo de geração e se baseia no seguinte princípio: dado um conjunto de uma população, no qual cada indivíduo está em determinada posição, o jogo se responsabiliza em gerar uma nova população baseada em determinadas regras.

O que difere um jogo da vida de outro são as regras. Existem diversas formas de decidir como será a próxima geração. As regras usadas no presente programa são:

* Qualquer célula viva com menos de dois vizinhos vivos morre de solidão
* Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação
* Qualquer célula morta com exatamente três vizinhos vivos se torna uma célula viva
* Qualquer célula viva com dois ou três vizinhos vivos continua no mesmo estado para a próxima geração.

O programa foi divido em três arquivos. O arquivo principal: **jogo\_da\_vida.c**, o arquivo que contém as funções que trabalham diretamente com matriz: **Matrix.h**, e o arquivo que contém funções utilitárias: **Util.h**.

**Implementação**

O programa segue um fluxo simples e bem segmentado, onde cada função (com um nome autoexplicativo) realiza um passo preparando os dados para a próxima ação, atingindo o objetivo no final.

* **Pega matriz do arquivo** (getMatrixByArq)

Composta por alguns passo, lê a matriz e preenche a struct.

* + **Valida arquivo** (testFileValid)

Se o arquivo for nulo, o programa se encerra.

* + **Aloca matriz** (allocMatrixByArq e allocMatrix)

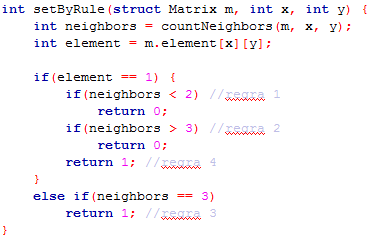
Lê a quantidade de colunas e linhas do arquivo, para saber o tamanho a ser alocado, e aloca, retornando a struct já alocada.

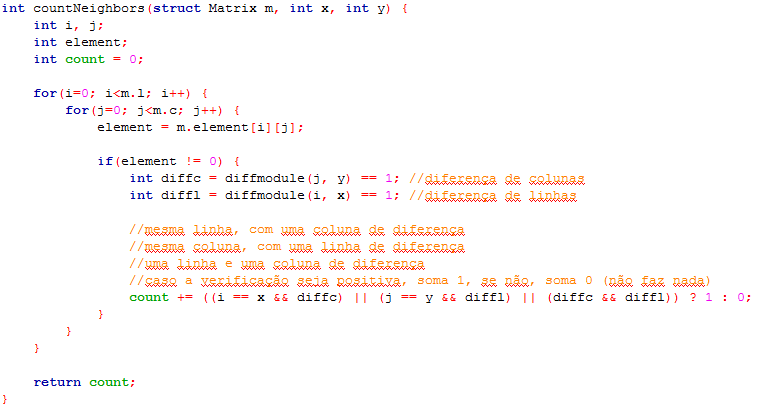
* + **Pega elementos do arquivo** (getElementsByArq)

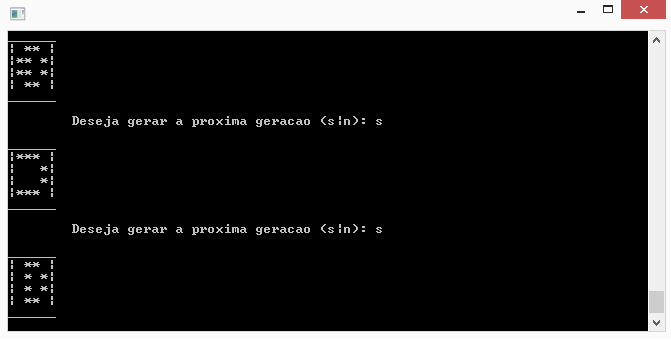
Percorre o arquivo e joga os elementos dentro da variável “element” da struct gerada anteriormente.

* **Imprime a matriz**
* **Verifica se usuário deseja gerar a próxima geração**

Caso negativo, salva a última matriz gerada no arquivo. Caso positivo, continua o fluxo do programa.

* **Gera a próxima população** (nextGeneration)
* Primeiramente é criada uma nova struct que contém a cópia exata da matriz atual, através da função **(copyMatrix)**.
* Depois, para cada elemento da matriz, é definido seu valor a partir das regras definidas para o jogo, através da função **(setByRule)**.
* A função **setByRule** se baseia na quantidade de vizinhos existentes para aquele ponto (countNeighbors) e nas regras definidas pelo jogo, como mostrado na figura acima.
* A função **countNeighbors** por sua vez conta a quantidade de vizinhos para um determinado ponto, como mostra a figura abaixo:





**Conclusão**

Acredito que o problema foi bem implementado, com um alto nível de modularização, facilitando tanto o desenvolvimento quanto o entendimento de terceiros.

Exemplo de funcionamento

A principal dificuldade encontrada foi ao fazer a função que retorna o número de vizinhos a partir de

um ponto específico da matriz.

Ela possui uma lógica razoavelmente simples, porém, meu desafio era implementá-la da forma mais otimizada o possível, e consegui um resultado satisfatório.

Foi criada uma variável count, que contém o número de vizinhos, e a cada elemento verificado, se ele estiver em uma posição considerada como vizinho, é somado 1 ao count, se não, é somado 0, ou seja, continua no mesmo estado.