> Vou jogar uma maldição em você e seus filhos vão nascer completamente pelados.
>
> Jimi Hendrix

1 Trabalhando com tabuleiros - o uso de matrizes

Em muitos casos, grandes jogos precisam de mais de uma dimensão para controle de seus dados. Exemplos são vários, como xadrez, dama, batalha naval e o jogo da velha. Por exemplo, um tabuleiro de xadrez possui 64 casas ou posições numeradas (figura 1).

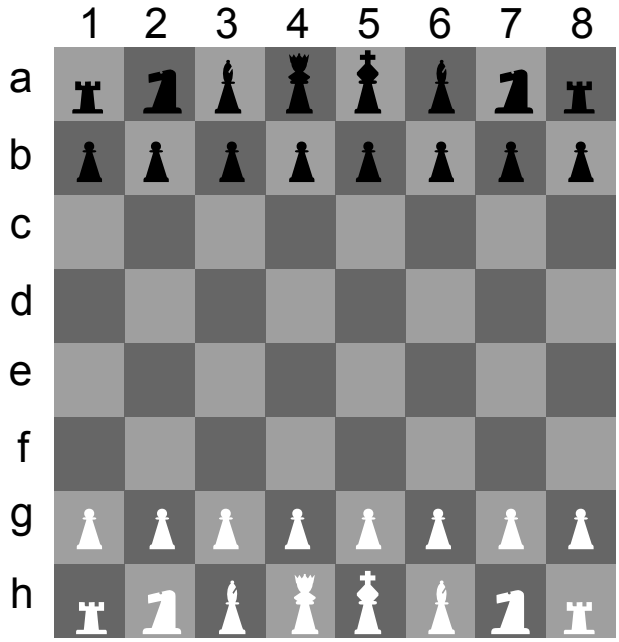


Figura 1: Um tabuleiro de xadrez.

Considerando a posição inicial do tabuleiro (figura 1) podemos realizar dois movimentos, um para cada jogador:

- Jogador com peças brancas move cavalo de H7 para F6;
- Jogador com peças pretas move peão de B3 para D3.

Após estas jogadas, a nova disposição do tabuleiro pode ser vista na figura 2.

1.1 E como definir um tabuleiro de xadrez em C++ ?

Felizmente, a linguagem C++ (e outras) permite representar situações como a do tabuleiro de xadrez de forma bem simples, que é utilizando matrizes.

Uma matriz é um arranjo bidimensional ou multidimensional de alocação estática e sequencial. A matriz é uma estrutura de dados que necessita de um índice para referenciar a linha e outro para referenciar a coluna para que seus

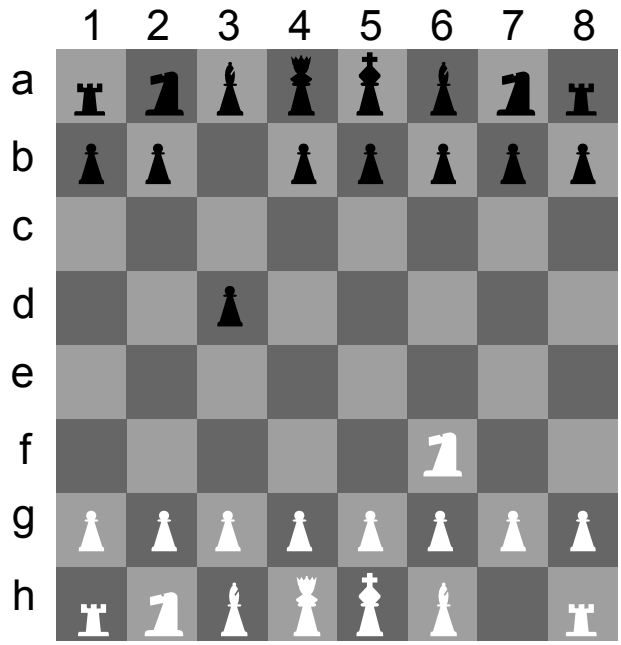


Figura 2: Um tabuleiro de xadrez após duas jogadas.

elementos sejam endereçados. Da mesma forma que um vetor, uma matriz é definido com um tamanho fixo, todas os elementos são do mesmo tipo, cada célula contém somente um valor. Os elementos ocupam posições contíguas na memória. Por compatibilidade com a matemática a primeira dimensão é chamada de linha e a segunda de colunas. A alocação dos elementos da matriz na memória pode ser feita colocando os elementos linha-por-linha ou coluna-por-coluna.

De forma geral, a declaração de vetores pode ser utilizada para definir matrizes n-dimensionais de dados. O vetor é um caso particular dessa declaração, no qual a matriz definida possui somente uma dimensão (linha). A forma geral da definição de matrizes em C++ é:

```
tipo nome_matriz[ quantidade_linhas ][ quantidade_colunas ];
```

O programa 1 ilustra, de forma simples, a declaração de uma matriz para representar o tabuleiro de xadrez e a realização das duas jogadas:

Programa 1: Movimentação das peças no tabuleiro.

```
//tabuleiro de xadrez
//programa_001.cpp
#include "biblaureano.h"

int main(){
   //declaração do tabuleiro -> 8x8 = 64 casas ;)
   string tabuleiroXadrez[8][8];

   //movendo cavalo branco para F6
   //onde F é representado pela sexta posição nas linhas
   //e 6 é a sexta posição nas colunas
   tabuleiroXadrez[5][5] = "cavalo branco";

   //movendo peão preto para D3
   //onde D é representado pela quarta posição na linhas
```

```
16      //e 3 é terceira posição nas colunas
17      tabuleiroXadrez[3][2] = "peão preto";
18      cout << "Game Over" << endl;
19      return 0;
20  }
```

A regra de acesso as linhas e colunas obedecem as mesmas regras dos vetores, ou seja, começam na posição 0.

2 Uma matriz de strings - brincando de embaralhar as frases

Um jogo simples de ser implementando é a adivinhação de palavras embaralhadas. O jogador fica tentando adivinhar a palavra que aparece e em algumas versões pode pedir dicas da palavra apresentada. O programa 2 apresenta uma versão deste jogo:

Programa 2: Movimentação das peças no tabuleiro.

```
1   //embaralhando as palavras
2   //programa_002.cpp
3   #include "biblaureano.h"
4
5   enum CAMPOS {PALAVRA, DICA, QTDCAMPOS};
6   const int NUM_PALAVRAS=5;
7   const string PALAVRAS[NUM_PALAVRAS][QTDCAMPOS] =
8       {
9           {"The Wall", "Música famosa..."},
10          {"Curitiba", "Capital."},
11          {"Monica", "Personagem de revista em quadrinhos."},
12          {"Jaspion", "herói japonês."},
13          {"Jogos Digitais", "curso de vocês."}
14      };
15
16  int main(){
17      int escolha = randomico(0,NUM_PALAVRAS);
18      string palavra = PALAVRAS[escolha][PALAVRA];
19      string dica = PALAVRAS[escolha][DICA];
20
21      string embaralhado = palavra;
22      int tamanho = embaralhado.size();
23      for( int i = 0; i<tamanho;++i){
24          int pos1 = randomico(0,tamanho);
25          int pos2 = randomico(0,tamanho);
26          char temp = embaralhado[pos1];
27          embaralhado[pos1] = embaralhado[pos2];
28          embaralhado[pos2] = temp;
29      }
30
31      cout << "Bem vindo ao jogo de palavras embaralhadas!" << endl;
32      do{
33          cout << "A palavra é:" << embaralhado << endl;
34          cout << "\tEntre com 'dica' para ver a dica." << endl;
35          cout << "\tEntre com 'fim' para finalizar." << endl;
36          cout << "\tOu entre com a palavra até acertar...." << endl;
37
38          string tentativa;
39          tentativa = readString("\tDigite a palavra:");
40
```

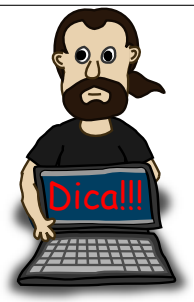
```
41          if( tentativa == palavra){
42              cout << "Parabéns..." << endl;
43              break;
44          }
45          else if( tentativa == "dica"){
46              cout << "A dica é " << dica << endl;
47          }
48          else if ( tentativa == "fim"){
49              cout << "By..by..." << endl;
50          }
51
52      } while(true);
53      cout << "Game over!!" << endl;
54      return 0;
55  }
```

2.1 Entendendo o programa

Inicialmente, criamos uma lista de palavras com dicas associadas:

```
1  ...
2  enum CAMPOS {PALAVRA, DICA, QTDCAMPOS};
3  const int NUM_PALAVRAS=5;
4  const string PALAVRAS[NUM_PALAVRAS][QTDCAMPOS] =
5      {
6          {"The Wall", "Música famosa..."},
7          {"Curitiba", "Capital."},
8          {"Monica", "Personagem de revista em quadrinhos."},
9          {"Jaspion", "herói japonês."},
10         {"Jogos Digitais", "curso de vocês."}
11     };
12 ...
```

Estamos declarando uma matriz com duas dimensões contendo as palavras e suas respectivas dicas. O *enum CAMPOS* define os enumeradores para acessar a matriz. Por exemplo, *PALAVRAS[x][PALAVRA]* contém a palavra que será embaralhada e *PALAVRAS[x][DICA]* a sua respectiva dica (onde x indica a posição da linha da matriz e *PALAVRA* e *DICA* indicam a coluna).



Uma bom método de definir a quantidade de enumeradores é criar um elemento ao final da lista para tal função. Lembrando que o primeiro enumerador definido conterá o valor 0 (zero). Por exemplo:

```
1  enum DIFICULDADE {FACIL, MEDIO, DIFICIL, QTD_NIVEIS}
2  cout << "O jogo tem " << QTD_NIVEIS << " de dificuldade" << endl;
```

No código anterior, QTD_NIVEIS está com o valor 3, o número exato de campos definido na enumeração.

O próximo passo é pegar randomicamente uma palavra e sua respectiva dica da matriz:

```
...
int escolha = randomico(0, NUM_PALAVRAS);
string palavra = PALAVRAS[escolha][PALAVRA];
string dica = PALAVRAS[escolha][DICA];
...
```

Na sequência, embaralhamos a frase repetidas vezes. O processo consiste em ficar trocando duas letras de lugar de randomicamente:

```
...
string embaralhado = palavra;
int tamanho = embaralhado.size();
for( int i = 0; i<tamanho;++i){
   int pos1 = randomico(0,tamanho);
   int pos2 = randomico(0,tamanho);
   char temp = embaralhado[pos1];
   embaralhado[pos1] = embaralhado[pos2];
   embaralhado[pos2] = temp;
}
...
```

E finalmente ficar questionando o usuário até que ele acerte a palavra embaralhada. O usuário tem a opção de pedir uma dica ou simplesmente desistir do jogo:

```
...
cout << "Bem vindo ao jogo de palavras embaralhadas!" << endl;
do{
     cout << "A palavra é:" << embaralhado << endl;
     cout << "\tEntre com 'dica' para ver a dica." << endl;
     cout << "\tEntre com 'fim' para finalizar." << endl;
     cout << "\tOu entre com a palavra até acertar...." << endl;

     string tentativa;
     tentativa = readString("Digite a palavra:");

     if( tentativa == palavra){
          cout << "Parabéns..." << endl;
          break;
     }
     else if( tentativa == 'dica'){
          cout << "A dica é " << dica << endl;
     }
     else if ( tentativa == 'fim'){
          cout << "By..by..." << endl;
     }

} while(true);
...
```

3 Lala - a barata tonta

Matrizes também podem ser utilizadas para representar cenários. Imagine que você tem uma cozinha e uma barata (bicho nojento) fique *passeando* pela cozinha. Claro que por onde a barata passar e deixar marcas das suas *patinhas* precisa ser limpo novamente. Então, a missão da barata *Lala* é percorrer toda a cozinha e sujá-la (figura 3). Mas *Lala* tem sérios problemas de sentido de direção e se esquece muito rápido por onde passou, então ela fica passando várias vezes no mesmo local. A única coisa que a barata *Lala* sabe é a quantidade de ladrilhos na cozinha e contar por quantos ladrilhos ela já passou, assim, precisamos ajudar a baratinha a *sujar* a cozinha. O programa 3 resolve o problema da barata *Lala*, pois ele indica direção que a barata deve seguir e ajuda a contar quantos ladrilhos da cozinha ela já sujou.

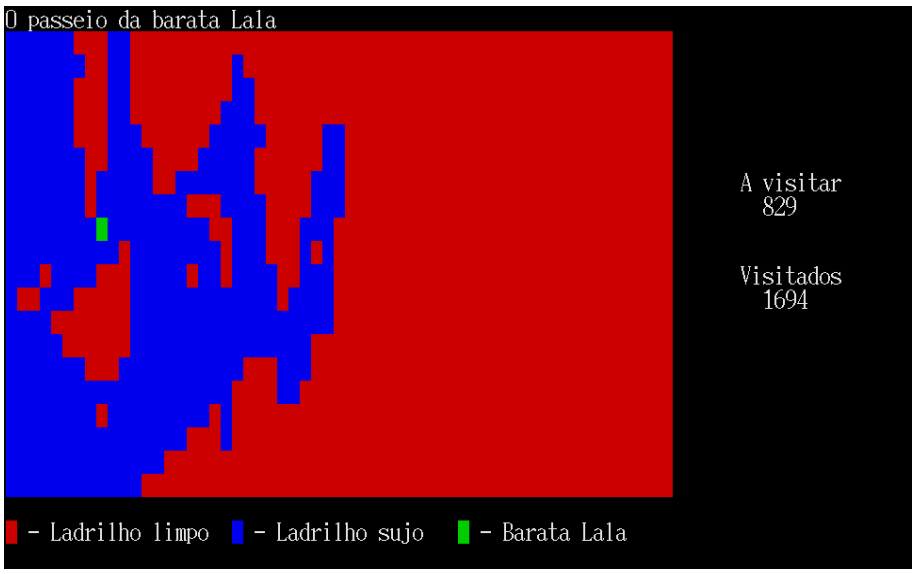


Figura 3: Lala - A barata tonta.

Programa 3: Barata passeando na cozinha.

```
1  //barata na cozinha
2  //adaptado do programa prof. Fabio Binder
3  //programa_003.cpp
4  #include "biblaureano.h"
5
6  const int CIMA = 0,
7            BAIXO = 1,
8            DIREITA = 2,
9            ESQUERDA = 3;
10
11 const int LARGURA = 20,
12           COMPRIMENTO = 60;
13
14 int main(){
15    int cozinha[LARGURA][COMPRIMENTO];
16    srand(time(0));
17
18    limparTela();
19    desligaCursor(true);
20    cout << "O passeio da barata Lala";
21
22    //informações
23    gotoXY(66,8);
24    cout << "A visitar";
25    gotoXY(66,12);
26    cout << "Visitados";
27
28    //legenda
29    gotoXY(2,23);
30    cout << " - Ladrilho limpo    - Ladrilho sujo     - Barata Lala" << endl;
31    gotoXY(0,23);
32    mudaCor(RED,RED);
33    cout << " ";
34    gotoXY(21,23);
35    mudaCor(BLUE,BLUE);
36    cout << " ";
37    gotoXY(41,23);
38    mudaCor(GREEN,GREEN);
39    cout << " ";
```

```
40
41      mudaCor(RED,RED);
42      for( int i = 0; i < LARGURA; ++i){
43          for( int j = 0; j < COMPRIMENTO; ++j){
44              cozinha[i][j] = 0; //matriz linha,coluna
45              gotoXY(j,i+2); //na tela é coluna,linha
46              cout << " ";
47          }
48      }
49      //barata anda
50      int visitados = 0, faltaVisitar = 0;
51      int linha=0,
52          coluna =0; //posição inicial da barata
53      while( faltaVisitar < LARGURA*COMPRIMENTO){
54          int direcao = randomico(0,4);
55          switch(direcao){
56              case CIMA:
57                  if( linha > 0) --linha;
58                  break;
59              case BAIXO:
60                  if( linha<LARGURA-1) ++linha;
61                  break;
62              case DIREITA:
63                  if( coluna < COMPRIMENTO-1) ++coluna;
64                  break;
65              case ESQUERDA:
66                  if( coluna > 0) --coluna;
67                  break;
68          }
69
70          if( cozinha[linha][coluna]++ == 0) faltaVisitar++;
71
72          mudaCor(GREEN,GREEN);
73          gotoXY(coluna,linha+2);
74          cout << "*";
75          limpaEfeito();
76          gotoXY(68,9);
77          cout << "      ";
78          gotoXY(68,9);
79          cout << LARGURA*COMPRIMENTO-faltaVisitar;
80          gotoXY(68,13);
81          cout << visitados;
82          espera(1);
83          mudaCor(BLUE,BLUE);
84          gotoXY(coluna,linha+2);
85          cout << " ";
86          ++visitados;
87      }
88      return 0;
89  }
```

3.1 Entendo o programa

Inicialmente definimos a dimensão da cozinha, no caso, representado por uma matriz de inteiros de *LARGURAxCOMPRIMENTO*:

```
1   ...
2   int cozinha[LARGURA][COMPRIMENTO];
3   ...
```

Na sequência, preenchemos toda a matriz com o valor 0 (zero) já montamos na tela a mesma matriz. Repare que a matriz é linha x coluna mas a tela é representado por coluna x linha, logo temos que inverter as coordenadas das linhas e colunas da matriz com a tela.

```
1   ...
2   for( int i = 0; i < LARGURA; ++i){
```

```
3       for( int j = 0; j < COMPRIMENTO; ++j){
4           cozinha[i][j] = 0;
5           gotoXY(j,i+2);
6           cout << " ";
7       }
8   }
9   ...
```

Enquanto a quantidade de ladrilhos a serem visitados não for atingido, o programa *define* o caminho que a barata *Lala* deve percorrer:

```
1   ...
2   int visitados = 0, faltaVisitar = 0;
3   int linha=0,
4       coluna =0; //posição inicial da barata
5   while( faltaVisitar < LARGURA*COMPRIMENTO){
6       int direcao = randomico(0,4);
7       switch(direcao){
8           case CIMA:
9               if( linha > 0) --linha;
10              break;
11          case BAIXO:
12              if( linha<LARGURA-1) ++linha;
13              break;
14          case DIREITA:
15              if( coluna < COMPRIMENTO-1) ++coluna;
16              break;
17          case ESQUERDA:
18              if( coluna > 0) --coluna;
19              break;
20      }
21      ...
```

Se o ladrinho não foi visitado, então ele contém o valor 0 (zero) que indica a quantidade de vezes que a barata *Lala* já passou por ali:

```
1       ...
2       if( cozinha[linha][coluna]++ == 0) faltaVisitar++;
3       ...
```

E finalmente, são apresentados os resultados do passeio da barata *Lala*:

```
1       ...
2       mudaCor(GREEN,GREEN);
3       gotoXY(coluna,linha+2);
4       cout << "*";
5       limpaEfeito();
6       gotoXY(68,9);
7       cout << "     ";
8       gotoXY(68,9);
9       cout << LARGURA*COMPRIMENTO-faltaVisitar;
10      gotoXY(68,13);
11      cout << visitados;
12      espera(1);
13      mudaCor(BLUE,BLUE);
14      gotoXY(coluna,linha+2);
15      cout << " ";
16      ++visitados;
17      ...
```

Neste programa utilizamos várias funções *cosméticas* como *mudaCor*, *gotoXY* e *limpaEfeito* apenas para melhorar o resultado final.

4 Um outro exemplo - simples e sem graça

Por exemplo, podemos definir uma matriz bidimensional de valores reais através da declaração em C++ que segue:

```
1   ...
2   float matrizReais[4][5];
3   ...
```

A variável *matrizReais* assim definida tem então 20 elementos (tabela 1).

Tabela 1: Matriz com 20 elementos

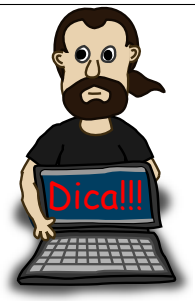
matrizReais[0][0]	matrizReais[0][1]	matrizReais[0][2]	matrizReais[0][3]	matrizReais[0][4]
matrizReais[1][0]	matrizReais[1][1]	matrizReais[1][2]	matrizReais[1][3]	matrizReais[1][4]
matrizReais[2][0]	matrizReais[2][1]	matrizReais[2][2]	matrizReais[2][3]	matrizReais[2][4]
matrizReais[3][0]	matrizReais[3][1]	matrizReais[3][2]	matrizReais[3][3]	matrizReais[3][4]

O acesso aos elementos da matriz *matrizReais* se dá de maneira similar ao acesso aos elementos de um vetor, ou seja, através de seus índices. O programa 4 apresenta um trecho de código em C++ para ler as dimensões e os elementos da matriz *matrizReais* acima, com 4 linhas e 5 colunas:

Programa 4: Leitura de dados de uma matriz.

```
1  //um exemplo sem glamour
2  //programa_004.cpp
3  #include "biblaureano.h"
4
5  int main(){
6     float matrizReais[4][5];
7
8     for( int i = 0; i < 4; ++i){
9        for( int j = 0; j < 5; ++j){
10          cout << "Entre com a linha "
11               << i+1
12               << " e coluna "
13               << j+1
14               << ":";
15          matrizReais[i][j] = readFloat();
16       }
17    }
```

```
18
19      cout << endl;
20      for( int i = 0; i < 4; ++i){
21          for( int j = 0; j < 5; ++j){
22              cout << matrizReais[i][j] << "\t";
23          }
24          cout << endl;
25      }
26
27      cout << "Game Over" << endl;
28      return 0;
29  }
```



Uma regra que pode-se sempre levar sempre em consideração: para cada dimensão de uma matriz, sempre haverá um laço de repetição (normalmente um *for*). Se houver 2 dimensões, então haverá 2 laços de repetição. Observe que na estrutura acima o laço externo percorre as linhas, e o laço interno percorre as colunas. Assim, para cada linha percorremos todas as colunas.

4.1 Matriz *n*-dimensionais

O conceito de dimensão pode ser estendido para mais de duas dimensões, criando-se matrizes *n*-dimensionais. Apesar de terem pouco uso prático deve-se lembrar que sempre cada dimensão definida irá ter o índice começando de zero e terminando em uma unidade antes do tamanho especificado para aquela dimensão.

Programa 5: Matriz de várias dimensões.

```
1  //outro exemplo sem glamour
2  //programa_005.cpp
3  #include "biblaureano.h"
4
5  #define DIMENSAO_1     2
6  #define DIMENSAO_2     5
7  #define DIMENSAO_3     3
8  #define DIMENSAO_4     4
9
10 int main(){
11     int matriz[DIMENSAO_1][DIMENSAO_2][DIMENSAO_3][DIMENSAO_4];
12
13     // Código para preencher uma Matriz de 4 dimensões
14     for (int i=0; i < DIMENSAO_1; ++i){
15        for (int j=0; j < DIMENSAO_2; ++j){
16           for (int k=0; k < DIMENSAO_3; ++k){
17              for (int l=0; l < DIMENSAO_4; ++l){
18                 matriz[i][j][k][l] = i+j+k+l;
19              }
20           }
21        }
22     }
```

```
23
24      // Uma regra que pode-se sempre levar sempre em consideração:
25      // para cada dimensão de uma matriz, sempre haverá um laço (normalmente um for).
26      // Se houver 4 dimensões, então haverá 4 laços.
27      for (int i=0; i < DIMENSAO_1; ++i){
28          for (int j=0; j < DIMENSAO_2; ++j){
29              for (int k=0; k < DIMENSAO_3; ++k){
30                  for (int l=0; l < DIMENSAO_4; ++l){
31                      cout << "Conteúdo da matriz Matriz["
32                           << i << "]["
33                           << j << "]["
34                           << k << "]["
35                           << l << "]:"
36                           << matriz[i][j][k][l] << endl;
37                  }
38              }
39          }
40      }
41
42      cout << "Game Over" << endl;
43      return 0;
44  }
```

5 Importância da utilização de informações matriciais em programas

As estruturas de dados vetoriais e matriciais são extremamente importantes em programas. Eis alguns exemplos de sua utilização:

- **Armazenar tabelas**: uma das utilizações mais frequentes de matrizes é o armazenamento de tabelas de dados, como a tabela de dados atmosféricos abaixo:

hora	temperatura	umidade	pressão
00	12.5	70	778
01	12.1	67	785
02	11.4	65	789
...	...	...	...
23	13.0	73	770

- **Tratamento de imagens**: uma imagem em um computador é geralmente representada por uma matriz bidimensional, onde cada elemento e_{xy} representa um *quadradinho* (pixel) da imagem. Desta forma, cada elemento *e[x,y]* da matriz irá conter a cor e/ou tonalidade dessa região. No exemplo apresentado na figura 4, *0* indica a cor branca e *1* a cor azul:

- **Cálculo numérico**: matrizes são geralmente usadas para a representação e resolução de sistemas de equações lineares normalmente encontrados em problemas de engenharia:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \cdots &= \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

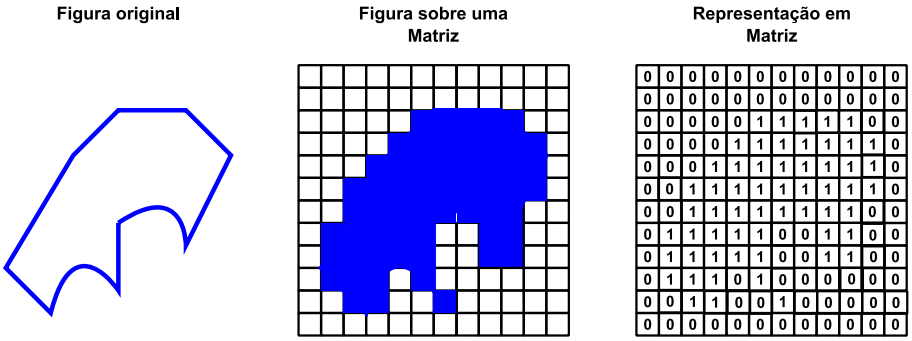


Figura 4: Representação de uma figura em uma matriz.

6 Tilesets - utilização de matrizes em jogos

Vários jogos utilizam uma técnica conhecida como *tile mapping*. Nessa técnica, cada elemento de um gráfico é representado por blocos menores que se encaixam. Assim, é possível gerar cenários de qualquer tamanho com poucos gráficos. O jogo *Free Civilization* (http://freeciv.wikia.com/) é um exemplo de jogo de estratégia que utiliza blocos para construir mapas (observe a figura 5).

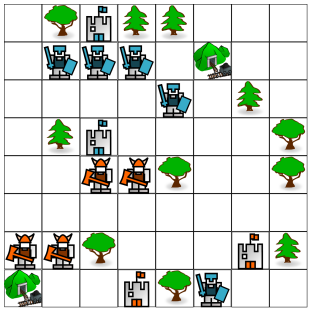
Figura 5: Um mapa feito com blocos - Exemplo do jogo *Free Civilization* (http://freeciv.wikia.com/wiki/Tilesets).

O uso de blocos para construir mapas nãos e aplica apenas a jogos de estratégia. Jogos como Zelda, Super Mario e outros possuem vários cenários (centenas) que utilizam a mesma técnica. O motivo para se utilizar *tiles* é economia de memória. Considere a figura 7 que apresenta um possível cenário para um jogo de estratégia. Como pode ser observado, o cenário é composto de 8 linhas por 8 colunas (64 blocos no total). Se considerarmos, hipoteticamente, que cada imagem tenha 1kb de memória, teríamos então um uso de 64kb utilizados (se não fosse utilizada a técnica de *tile mapping*). Mas o mapa apresentado contém apenas 8 elementos gráficos diferentes (figura 6).

Portanto, se utilizarmos uma matriz contendo os índices (referências) para cada elemento gráfico (6, podemos representar o mesmo cenário com significativa economia de memória. O programa 6 demonstra uma possível representação para o cenário apresentado.

Programa 6: Exemplo da matriz para o jogo de estratégia.

Figura 6: Elementos gráficos do jogo de estratégia.



1	2	7	3	3	1	1	1
1	6	6	6	1	4	1	1
1	1	1	1	6	1	3	1
1	3	7	1	1	1	1	2
1	1	5	5	2	1	1	2
1	1	1	1	1	1	1	1
5	5	2	1	1	1	8	3
4	1	1	8	2	6	1	1

Figura 7: Um exemplo com jogo de estratégias.

```
//tilesets
//programa_006.cpp
#include "biblaureano.h"

#define LINHAS 8
#define COLUNAS 8

const char VAZIO = '.', //bloco vazio
       PLANTA_1 = 'p', //primeira árvore
       PLANTA_2 = 'x', //segunda árvore
       MINA = 'M', //mina
       SOLDADO_1 = 'L', //soldado laranja
       SOLDADO_2 = 'A', //soldado azul
       CASTELO_1 = 'l', //castelo laranja
       CASTELO_2 = 'a'; //castelo azul

int main(){
   char cenario[LINHAS][COLUNAS] =
   {
      {VAZIO,     PLANTA_1, CASTELO_1,PLANTA_2, PLANTA_2, VAZIO,     VAZIO,     VAZIO},
      {VAZIO,     SOLDADO_2,SOLDADO_2,SOLDADO_2,VAZIO,    MINA,      VAZIO,     VAZIO},
      {VAZIO,     VAZIO,    VAZIO,    VAZIO,    SOLDADO_2,VAZIO,     PLANTA_2,  VAZIO},
      {VAZIO,     PLANTA_2, CASTELO_2,VAZIO,    VAZIO,    VAZIO,     VAZIO,     PLANTA_1},
      {VAZIO,     VAZIO,    SOLDADO_1,SOLDADO_1,PLANTA_1, VAZIO,     VAZIO,     PLANTA_1},
      {VAZIO,     VAZIO,    VAZIO,    VAZIO,    VAZIO,    VAZIO,     VAZIO,     VAZIO},
      {SOLDADO_1,SOLDADO_1,PLANTA_1,  VAZIO,    VAZIO,    VAZIO,     CASTELO_1,PLANTA_2},
      {MINA,      VAZIO,    VAZIO,    CASTELO_1,PLANTA_1, SOLDADO_2,VAZIO,     VAZIO}
   };

   for( int linhas = 0; linhas < LINHAS; ++linhas){
      for( int colunas = 0; colunas < COLUNAS; ++colunas){
         limpaEfeito();
         switch(cenario[linhas][colunas]){
            case PLANTA_1:
            case PLANTA_2:
            case MINA:
               mudaCor(GREEN);
               break;
            case CASTELO_1:
            case SOLDADO_1:
```

```
41                      mudaCor(YELLOW);
42                      break;
43                  case CASTELO_2:
44                  case SOLDADO_2:
45                      mudaCor(BLUE);
46                      break;
47              }
48              cout << cenario[linhas][colunas];
49          }
50          cout << endl;
51      }
52
53      cout << "Game Over" << endl;
54      return 0;
55  }
```

7 Matrizes na matemática

Na matemática, uma matriz é uma tabela de $m \times n$ símbolos. As linhas horizontais da matriz são chamadas de linhas e as linhas verticais são chamadas de colunas. Uma matriz com m linhas e n colunas é chamada de uma matriz m por n (escreve-se $m \times n$) e m e n são chamadas de suas dimensões, tipo ou ordem.

Um elemento de uma matriz A que está na i-ésima linha e na j-ésima coluna é chamado de elemento i,j ou (i,j)-ésimo elemento de A. Ele é escrito como $a_{i,j}$ ou $a[i,j]$ ou a_{ij} [1]. A figura 8 ilustra o conceito de matriz na matemática.

Uma matriz onde uma de suas dimensões é igual a 1 é geralmente chamada de **vetor**. Uma matriz $1 \times n$ (uma linha e n colunas) é chamada de vetor linha ou matriz linha, e uma matriz $m \times 1$ (uma coluna e m linhas) é chamada de vetor coluna ou matriz coluna.

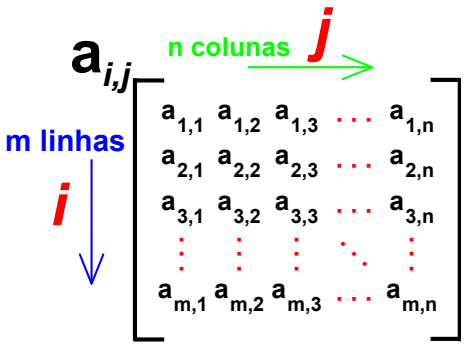


Figura 8: Representação de uma matriz na matemática.

7.1 Cálculo de determinantes

Em matemática, determinante é uma função que associa a cada matriz quadrada (mesmo número de linhas e colunas) um escalar. Esta função permite saber se a matriz tem ou não inversa, pois as que não têm são precisamente aquelas cujo determinante é igual a 0.

[1] Formato adotado no restante do texto

Determinante de uma matriz de ordem 1 : O determinante da matriz M de ordem $n = 1$, é o próprio número que origina a matriz. Dada uma matriz quadrada de 1ªordem $M = \begin{bmatrix} a_{11} \end{bmatrix}$. temos que o determinante é o número real a_{11}.

Determinante de uma matriz de ordem 2 : O determinante de uma matriz de 2ªordem é a diferença entre o produto dos termos da diagonal principal e o produto dos termos da diagonal secundária. Esses produtos se chamam, respectivamente, termo principal e termo secundário da matriz. Onde:

$$\det M = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11} * a_{22} - a_{12} * a_{21}.$$

Determinante de uma matriz de ordem 3 : Para calcular o determinante de matrizes de 3ªordem, utilizamos a chamada regra de *Sarrus*, que resulta no seguinte cálculo:

$$\det M = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & a_{31} & a_{32} \end{bmatrix}$$

Que resulta na expressão linear:

$$[(a_{11} * a_{22} * a_{33}) + (a_{12} * a_{23} * a_{31}) + (a_{13} * a_{21} * a_{32})] - [(a_{13} * a_{22} * a_{31}) + (a_{11} * a_{23} * a_{32}) + (a_{12} * a_{21} * a_{33})]$$

O programa 7 demonstra, em C++, como realizar o cálculo das determinantes de 1ª, 2ªe 3ªordem:

Programa 7: Cálculo de determinante de matrizes.

```
1  //cálculo determinante
2  //programa_007.cpp
3  #include "biblaureano.h"
4
5  int main(){
6     //declaração da matriz
7     float matrizPrimeiraOrdem[1][1];
8
9     matrizPrimeiraOrdem[0][0] = readFloat("Entre com o valor da posição [0][0]:");
10
11    //cálculo determinante de matriz de primeira ordem
12    float determinante = matrizPrimeiraOrdem[0][0];
13    cout << "Determinante da matriz é:" << determinante << endl;
14
15    //declaração da matriz
16    float matrizSegundaOrdem[2][2];
17
18    //leitura da matriz
19    for( int i=0; i<2; ++i){
20        for( int j=0;j<2;++j){
21           cout << "Entre com o valor da posição ["
22                << i << "][" << j << "]:";
23           matrizSegundaOrdem[i][j] = readFloat();
24        }
25    }
26
27    //cálculo do determinante
28    determinante = (matrizSegundaOrdem[0][0] * matrizSegundaOrdem[1][1])-
29                   (matrizSegundaOrdem[0][1] * matrizSegundaOrdem[1][0]);
30    cout << "Determinante da matriz é:" << determinante << endl;
31
32    //declaração da matriz
33    float matrizTerceiraOrdem[3][3];
34
35    //leitura da matriz
36    for( int i=0; i<3; ++i){
37        for( int j=0;j<3;++j){
38           cout << "Entre com o valor da posição ["
39                << i << "][" << j << "]:";
40           matrizTerceiraOrdem[i][j] = readFloat();
41        }
```

```
42      }
43
44      //cálculo do determinante
45      determinante =
46          ((matrizTerceiraOrdem[0][0]*matrizTerceiraOrdem[1][1]*matrizTerceiraOrdem[2][2])+
47           (matrizTerceiraOrdem[0][1]*matrizTerceiraOrdem[1][2]*matrizTerceiraOrdem[2][0])+
48           (matrizTerceiraOrdem[0][2]*matrizTerceiraOrdem[1][0]*matrizTerceiraOrdem[2][1]))
49            -
50          ((matrizTerceiraOrdem[0][2]*matrizTerceiraOrdem[1][1]*matrizTerceiraOrdem[2][0])+
51           (matrizTerceiraOrdem[0][0]*matrizTerceiraOrdem[1][2]*matrizTerceiraOrdem[2][1])+
52           (matrizTerceiraOrdem[0][1]*matrizTerceiraOrdem[1][0]*matrizTerceiraOrdem[2][2]));
53
54      cout << "Determinante da matriz é:" << determinante << endl;
55
56      cout << "Game Over" << endl;
57      return 0;
58  }
```

8 Outro jogo de tabuleiro - jogo da velha

O jogo da velha deve ser um dos primeiros jogos de tabuleiros que aprendemos (se não for o primeiro). O objetivo do jogo é simples: preencher na horizontal, vertical ou diagonal uma linha com o seu símbolo, normalmente representando por um O ou um X (observe a figura 9).

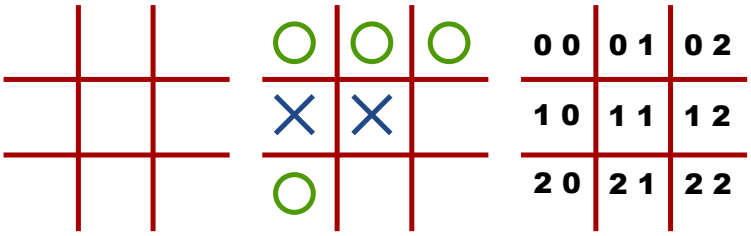


Figura 9: Jogo da velha.

Observando a sequência apresentada na figura 9 podemos concluir que temos um jogo simples composto de três linhas e três colunas, portanto temos uma matriz. Este tabuleiro pode ser facilmente representando numa linguagem de programação. Considerando a sequência de jogadas apresentadas na figura 9, esta matriz pode ser representada em C++ pelo código:

```
1   ...
2   //declaração de uma matriz 3x3
3   const int LINHAS=3;
4   const int COLUNAS=3;
5   char velha[LINHAS][COLUNAS] = {{ 'O', 'O', 'O'},
6                                  { 'X', 'X', ' '},
7                                  { 'O', ' ', ' '}};
8   ...
```

Um exemplo mais completo das jogadas, inclusive o acesso a cada posição da matriz, pode ser visto no programa 8.

Programa 8: Um exemplo com jogo da velha.

```cpp
//jogo da velha
//programa_008.cpp
#include "biblaureano.h"

int main(){
   //declaração de uma matriz 3x3
   const int LINHAS=3;
   const int COLUNAS=3;
   char velha[LINHAS][COLUNAS] = {{ ' ', 'O', 'O'},
                                  { 'X', 'X', ' '},
                                  { 'O', ' ', ' '}};

   cout << "Tabuleiro atual" << endl;
   for( int i=0;i<LINHAS;++i){
      for( int j=0; j<COLUNAS;++j){
         cout << velha[i][j];
      }
      cout << endl;
   }

   cout << "Movendo 'O' para uma posição vazia." << endl;
   velha[0][0] = 'O';

   cout << "Tabuleiro atual" << endl;
   for( int i=0;i<LINHAS;++i){
      for( int j=0; j<COLUNAS;++j){
         cout << velha[i][j];
      }
      cout << endl;
   }

   cout << "Jogador 'O' ganhou!!" << endl;
   cout << "Game Over" << endl;
   return 0;
}
```

Considere o seguinte problema: criar um jogo da velha para 2 jogadores. O jogo deve informar quando houver jogadas inválidas, considerar 3 o tamanho do tabuleiro. O programa 9 apresenta resposta para este problema.

Programa 9: Um exemplo com jogo da velha.

```cpp
//jogo da velha completo
//programa_009.cpp
#include "biblaureano.h"

int main(){
   //declaração de uma matriz 3x3
   const int LINHAS=3;
   const int COLUNAS=3;
   char velha[LINHAS][COLUNAS] = {{ ' ', ' ', ' '},
                                  { ' ', ' ', ' '},
                                  { ' ', ' ', ' '}};

   char jogador = 'X';
   bool houveGanhador = false;
   int qtdJogadas = 0;

   //enquanto não houve ganhador e quantidade de jogadas
   // for menor que 9 (3x3)
   while( houveGanhador == false && qtdJogadas < 9){
      //montando o tabuleiro
      cout << "Tabuleiro atual" << endl;
      for( int i=0;i<LINHAS;++i){
         for( int j=0; j<COLUNAS;++j){
            cout << velha[i][j];
         }
         cout << endl;
      }

      //alterna o jogador (lembre-se que este é um if simplificado)
      jogador = (jogador == 'X' ? 'O':'X');
```

```
31
32      int linha, coluna;
33      do{
34         cout << "Atenção jogador " << jogador << endl;
35         linha = readInt("\tEntre com a linha sua jogada:");
36         coluna = readInt("\tEntre com a coluna sua jogada:");
37      } while( linha<0  || linha>=LINHAS ||
38               coluna<0 || coluna>=COLUNAS || velha[linha][coluna]!=' ');
39
40      velha[linha][coluna] = jogador;
41
42      //verifica se houve ganhador
43      // verificação por linha
44      for( int i=0;i<LINHAS;++i){
45         if( velha[ i ][ 0 ] == jogador &&
46             velha[ i ][ 1 ] == jogador &&
47             velha[ i ][ 2 ] == jogador ){
48            houveGanhador = true;
49         }
50      }
51
52      //verifica se houve ganhador
53      // verificação por coluna
54      for( int j=0;j<LINHAS;++j){
55         if( velha[ 0 ][ j ] == jogador &&
56             velha[ 1 ][ j ] == jogador &&
57             velha[ 2 ][ j ] == jogador ){
58            houveGanhador = true;
59         }
60      }
61
62      //verifica as duas diagonais
63      if( (velha[0][0] == jogador &&
64           velha[1][1] == jogador &&
65           velha[2][2] == jogador) ||
66          (velha[0][2] == jogador &&
67           velha[1][1] == jogador &&
68           velha[2][0] == jogador)){
69         houveGanhador = true;
70      }
71      //incrementa a quantidade de jogadas
72      ++qtdJogadas;
73   }
74
75   if( houveGanhador == true ){
76       cout << "Jogador "
77            << jogador
78            << " ganhou o jogo!" << endl;
79   }
80   else{
81       cout << "Não houve vencedores!!" << endl;
82   }
83   cout << "Game Over" << endl;
84   return 0;
85 }
```

8.1 Entendendo o programa

De forma idêntica a declaração de um vetor, na declaração de uma matriz é possível inicializar a mesma com algum valor. Só devemos tomar o cuidado de fazer a separação entre { } (chaves).

```
1   ...
2   char velha[LINHAS][COLUNAS] = {{ ' ', ' ', ' '},
3                                  { ' ', ' ', ' '},
4                                  { ' ', ' ', ' '}};
5   ...
```

Na sequência, declara-se algumas variáveis que serão utilizadas durante o jogo e garantimos que o jogo prossiga até que haja um ganhador ou que tenha ocorrido 9 jogadas.

```
...
char jogador = 'X';
bool houveGanhador = false;
int qtdJogadas = 0;

//enquanto não houve ganhador e quantidade de jogadas
// for menor que 9 (3x3)
while( houveGanhador == false && qtdJogadas < 9){
    //outros comandos aqui
    ...
}
...
```

Para a impressão do tabuleiro, o programa percorre todas as colunas (segundo comando *for*) de cada linha (primeiro comando *for*).

```
    ...
    //montando o tabuleiro
    cout << "Tabuleiro atual" << endl;
    for( int i=0;i<LINHAS;++i){
        for( int j=0; j<COLUNAS;++j){
            cout << velha[i][j];
        }
        cout << endl;
    }
    ...
```

Utilizando o operador *? :*, implementamos de forma simples um *if..else*.

```
    ...
    //alterna o jogador (lembre-se que este é um if simplificado)
    jogador = (jogador == 'X' ? 'O':'X');
    ...
```

A linguagem C++ fornece um operador condicional *? :* (também chamado de operador ternário) que é muito semelhante a um bloco *if..else*. Este operador aceita três operandos. O primeiro operando é uma condição, o segundo é valor para a expressão condicional se o resultado do teste for *true*. O terceiro operando é o valor para a expressão condicional caso o resultado do teste for *false*.
Apesar de possuir a mesma funcionalidade não se deve usar este operador quando os comandos envolvidos são complexos. Primeiramente a condição é avaliada. Dependendo do resultado o bloco respectivo será executado. No caso a linha:

```
1       jogador = (jogador == 'X' ? 'O':'X');
```

É equivalente a escrever:

```
1     if( jogador == 'X'){
2        jogador = 'O';
3     }
4     else{
5        jogador = 'X';
6     }
```

O jogo deve garantir que não seja informado nenhuma coordenada inválida pelo jogador e que o jogador não consiga preencher uma casa que já tenha sido utilizado.

```
1       ...
2       int linha, coluna;
3       do{
4          cout << "Atenção jogador " << jogador << endl;
5          linha = readInt("\tEntre com a linha sua jogada:");
6          coluna = readInt("\tEntre com a coluna sua jogada:");
7       } while( linha<0  || linha>=LINHAS ||
8                coluna<0 || coluna>=COLUNAS || velha[linha][coluna]!=' ');
9       ...
```

Após ter passado pela validação, a jogada é realizada:

```
1     velha[linha][coluna] = jogador;
```

Para depois verificarmos se o jogador ganhou ou não aquele jogo. A primeira verificação ocorre fixando as colunas e alterando as linhas (verificação horizontal):

```
1       ...
2       // verificação por linha
3       for( int i=0;i<LINHAS;++i){
4          if( velha[ i ][ 0 ] == jogador &&
5              velha[ i ][ 1 ] == jogador &&
6              velha[ i ][ 2 ] == jogador ){
7             houveGanhador = true;
```

```
8           }
9       }
10      ...
```

A segunda verificação ocorre fixando as linhas e alterando as colunas (verificação na vertical):

```
1       ...
2       // verificação por coluna
3       for( int j=0;j<LINHAS;++j){
4           if( velha[ 0 ][ j ] == jogador &&
5               velha[ 1 ][ j ] == jogador &&
6               velha[ 2 ][ j ] == jogador ){
7              houveGanhador = true;
8           }
9       }
10      ...
```

Finalmente, a terceira verificação ocorre nas diagonais.

```
1       ...
2       //verifica as duas diagonais
3       if( (velha[0][0] == jogador &&
4            velha[1][1] == jogador &&
5            velha[2][2] == jogador) ||
6           (velha[0][2] == jogador &&
7            velha[1][1] == jogador &&
8            velha[2][0] == jogador)){
9          houveGanhador = true;
10      }
11      ...
```

Após a verificação é incrementado a quantidade de jogadas e o programa retorna ao início (comando *while*).

```
1       ...
2       //incrementa a quantidade de jogadas
3       ++qtdJogadas;
4       ...
```

No final do programa, é verificado se houve ou não algum ganhador e mostrado a mensagem adequada.

```
1   ...
2   if( houveGanhador == true ) {
3       cout << "Jogador "
4            << jogador
5            << " ganhou o jogo!" << endl;
6   }
7   else{
8       cout << "Não houve vencedores!!" << endl;
9   }
10  ...
```

Como este é um comando de seleção simples, podemos utilizar o operador ternário (?:) para verificação da condição e exibição das mensagens. O operador ternário (?:) poderia ser aplicado:

```
houveGanhador == true ?
                    cout << "Jogador " << jogador << " ganhou o jogo!" << endl :
                    cout << "Não houve vencedores!!" << endl;
```

Como pode ser percebido, neste caso o comando fica *confuso* devido a grande quantidade de comandos e portanto recomenda-se a utilização dos comandos *if..else* por questões de *legibilidade* do código escrito.

9 Batalha naval - do papel para o computador

Batalha naval é um jogo de tabuleiro de dois jogadores, no qual os jogadores têm de adivinhar em que quadrados estão os navios do oponente. Embora existam várias versões em tabuleiro ou eletrônicos, o jogo foi originalmente jogado com lápis e papel.

O jogo original é jogado em duas tabelas para cada jogador - uma que representa a disposição dos barcos do jogador, e outra que representa a do oponente. As tabelas são tipicamente quadradas, estando identificadas na horizontal por números e na vertical por letras. Em cada tabela o jogador coloca os seus navios e regista os tiros do oponente.

Antes do início do jogo, cada jogador coloca os seus navios (figura 10) nos quadros, alinhados horizontalmente ou verticalmente. O número de navios permitidos é igual para ambos jogadores e os navios não podem se sobrepor.

Após os navios terem sido posicionados (figura 11) o jogo continua numa série de turnos, em cada turno um jogador diz um quadrado na tabela do oponente, se houver um navio nesse quadrado, é colocada uma marca vermelha, senão houver é colocada uma marca branca.

Observando a figura 11 percebemos que novamente o tabuleiro é representando por uma matriz (15x15). Na versão em papel, o jogo é composto de (figura 12):

- 5 Hidroaviões
- 4 Submarinos
- 3 Cruzadores
- 2 Encouraçados
- 1 Porta-aviões

Distribuídos em uma tabela quadriculada de 15x15 (uma matriz).

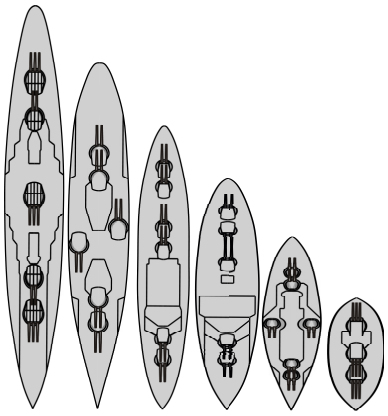


Figura 10: Um tabuleiro de batalha naval.

Para a preparação do jogo é necessário:

1. Cada jogador distribui suas embarcações pelo tabuleiro. Isso é feito marcando-se os quadradinhos referentes às suas armas.
2. Não é permitido que uma embarcação fique em cima de outra.
3. O jogador não deve revelar ao oponente as localizações de suas embarcações.

E as regras para jogar são fáceis de serem seguidas, pois cada jogador, na sua vez de jogar, seguirá o seguinte procedimento:

1. Disparará 1 tiro, indicando a coordenadas do alvo através do número da linha e da letra da coluna que definem a posição. Para que o jogador tenha o controle dos tiros disparados, deverá marcar cada um deles no seu jogo.
2. Após cada tiro, o oponente avisará se acertou e, nesse caso, qual a arma foi atingida. Se ela for afundada, esse fato também deverá ser informado.
3. A cada tiro acertado em um alvo, o oponente deverá marcar em seu tabuleiro para que possa informar quando a arma for afundada.
4. Uma arma é afundada quando todas as casas que formam essa arma forem atingidas.
5. Após o tiro e a resposta do oponente, a vez passa para o outro jogador.

A finalização do jogo ocorre quando um dos jogadores afundar todas as embarcações do seu oponente.

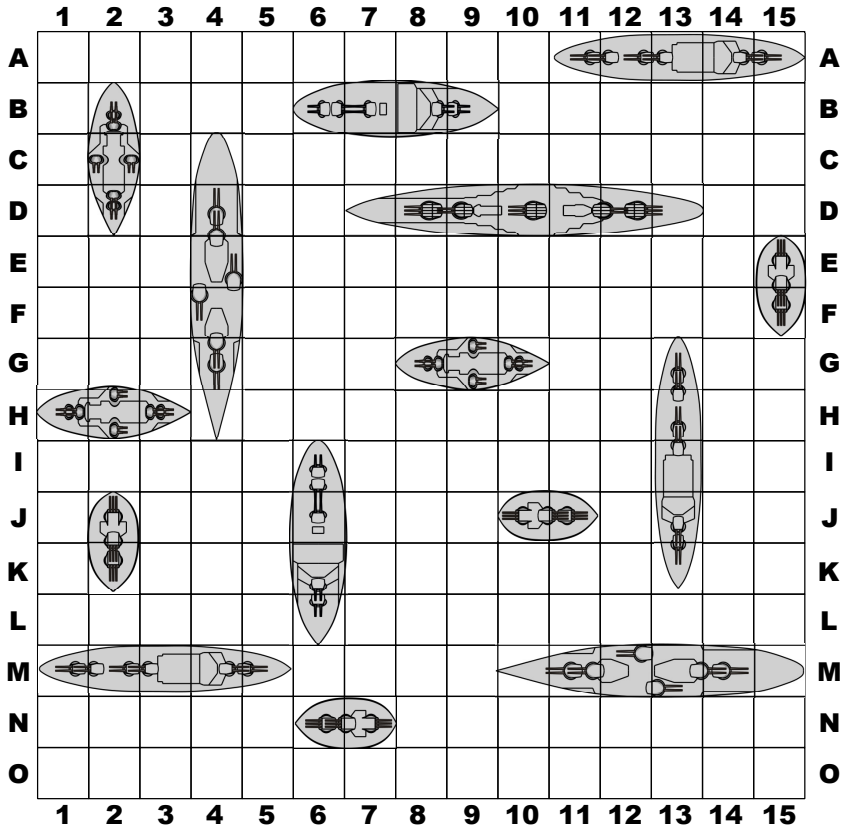


Figura 11: Um tabuleiro de batalha naval.

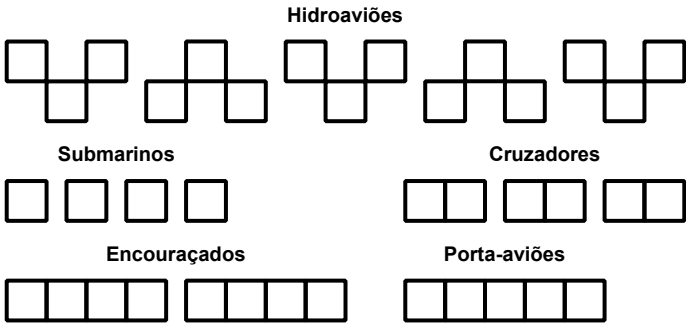


Figura 12: Embarcações da versão em papel.

O programa 10 apresenta uma implementação simples do jogo de batalha naval em C++:

Programa 10: Montando um tabuleiro simples de batalha naval.

```
1  //batalha naval
2  //programa_010.cpp
3  #include "biblaureano.h"
4
5  #define DIMENSAO 15 //dimensão do tabuleiro
6  int main()
7  {
8     //declaração da matriz
9     char batalhaNaval[DIMENSAO][DIMENSAO];
10
11    //preenchendo todo o tabuleiro com "água"
12    for( int i=0; i<DIMENSAO; ++i){
13       for( int j=0; j<DIMENSAO; ++j){
14          batalhaNaval[i][j] = '.';
15       }
16    }
17
18    //distribuindo todos os navios
19    //1 portaviões
20    batalhaNaval[6][7]='P';
21    batalhaNaval[6][8]='P';
22    batalhaNaval[6][9]='P';
23    batalhaNaval[6][10]='P';
24    batalhaNaval[6][11]='P';
25
26    //4 submarinos
27    batalhaNaval[7][1]='S';
28    batalhaNaval[9][14]='S';
29    batalhaNaval[0][4]='S';
30    batalhaNaval[11][5]='S';
31
32    //3 cruzadores
33    batalhaNaval[9][1]='C';
34    batalhaNaval[10][1]='C';
35    batalhaNaval[13][11]='C';
36    batalhaNaval[14][11]='C';
37    batalhaNaval[7][4]='C';
38    batalhaNaval[7][5]='C';
39
40    //2 encouraçados
41    batalhaNaval[0][9]='E';
42    batalhaNaval[1][9]='E';
43    batalhaNaval[2][9]='E';
44    batalhaNaval[3][9]='E';
45
46    batalhaNaval[13][2]='E';
47    batalhaNaval[13][3]='E';
48    batalhaNaval[13][4]='E';
49    batalhaNaval[13][5]='E';
50
51    //5 hidro-aviões
52    batalhaNaval[0][0]='H';
53    batalhaNaval[1][1]='H';
54    batalhaNaval[2][0]='H';
55
56    batalhaNaval[3][5]='H';
57    batalhaNaval[4][4]='H';
58    batalhaNaval[5][5]='H';
59
60    batalhaNaval[11][11]='H';
61    batalhaNaval[10][10]='H';
62    batalhaNaval[11][9]='H';
63
64    batalhaNaval[4][12]='H';
65    batalhaNaval[5][13]='H';
66    batalhaNaval[4][14]='H';
67
68    batalhaNaval[9][6]='H';
69    batalhaNaval[8][7]='H';
```

```
70      batalhaNaval[9][8]= 'H';
71
72      //imprimindo o tabuleiro
73      for( int i=0; i<DIMENSAO; ++i){
74          for( int j=0; j<DIMENSAO; ++j){
75              cout << batalhaNaval[i][j];
76          }
77          cout << endl;
78      }
79
80      cout << "Game Over" << endl;
81      return 0;
82  }
```

10 Outros exemplos matemáticos

As matrizes são muito utilizadas na computação para representarmos translação, rotação, escala de objetos em computação gráfica, para se resolver sistemas de equações, etc. Na engenharia elétrica, é muito difícil resolver problemas de circuitos elétricos e linhas de transmissão de energia elétrica sem matrizes. Trabalhar com uma malha de linha de transmissão e passar esse circuito para forma matricial, mais fácil. Na mecânica também é muito importante, pois os tensores (grandeza) só são fornecidos em forma de matriz. Devido a importância da utilização de matrizes, veremos mais algumas aplicações matemáticas com matrizes.

10.1 Transposta de uma matriz

Matriz transposta, em matemática, é o resultado da troca de linhas por colunas em uma determinada matriz. A matriz transposta de uma matriz qualquer M é representada por M^t. Onde:

$$M = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \rightarrow M^t = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{33} \end{bmatrix}$$

O programa 11 demonstra, em C++, como obter a transposta de uma matriz:

Programa 11: Cálculo de matriz transposta.

```
1   //matriz transposta
2   //programa_011.cpp
3   #include "biblaureano.h"
4
5   #define DIMENSAO 15 //dimensão do tabuleiro
6   int main(){
7       //matriz de ordem 3x2
8       int matriz3x2[3][2];
9       //matriz de ordem 2x3
10      int matrizTransposta[2][3];
11
12      //leitura dos dados das matrizes
13      for( int i=0;i<3;++i){
14          for( int j=0; j<2;++j){
15              cout << "Entre com o elemento "
16                   << i << "," << j << " da matriz:";
17              matriz3x2[i][j] = readInt();
18          }
19      }
20
21      //cálculo da transposta
22      for( int i=0;i<3;++i){
23          for( int j=0; j<2;++j){
```

```
24          //aqui ocorre a inversão
25          matrizTransposta[j][i] = matriz3x2[i][j];
26       }
27    }
28
29    //impressão
30    cout << "matriz Original:" << endl;
31    for( int i=0;i<3;++i){
32       for( int j=0; j<2;++j){
33          cout << matriz3x2[i][j] << "\t";
34       }
35       cout << endl;
36    }
37
38    cout << "matriz Transposta:" << endl;
39    for( int i=0;i<2;++i){
40       for( int j=0; j<3;++j){
41          cout << matrizTransposta[i][j] << "\t";
42       }
43       cout << endl;
44    }
45
46    cout << "Game Over" << endl;
47    return 0;
48 }
```

10.2 Adição de matrizes

A adição de matrizes só pode ocorrer se e somente se as matrizes forem de mesma ordem, ou seja, tenham o mesmo número de linhas e de colunas. Para fazer a adição entre os elementos dessa matriz basta somar os termos correspondente de ambas, $a_{ij} + b_{ij} = c_{ij}$, que resultará em uma terceira matriz. Temos de exemplo a soma das matrizes A e B de ordem $m \times n$ essa soma resultará na matriz C de ordem $m \times n$.

O programa 12 demonstra, em C++, como somar 2 matrizes:

Programa 12: Adição de matrizes.

```
1  //soma de matrizes
2  //programa_012.cpp
3  #include "biblaureano.h"
4
5  #define DIMENSAO_MAX 100 //dimensão máxima da matriz
6
7  int main(){
8     int matrizA[DIMENSAO_MAX][DIMENSAO_MAX];
9     int matrizB[DIMENSAO_MAX][DIMENSAO_MAX];
10    int matrizSoma[DIMENSAO_MAX][DIMENSAO_MAX];
11
12    int dimensaoI, dimensaoJ;
13
14    do{
15       dimensaoI = readInt("Entre com dimensão I das matrizes:");
16       dimensaoJ = readInt("Entre com dimensão J das matrizes:");
17    } while( dimensaoI <=0 || dimensaoI >= DIMENSAO_MAX ||
18             dimensaoJ <=0 || dimensaoJ >= DIMENSAO_MAX );
19
20    //leitura dos dados das matrizes
21    cout << "Leitura da matriz A:"<<endl;
22    for( int i=0;i<dimensaoI;++i){
23       for( int j=0; j<dimensaoJ;++j){
24          cout << "Entre com o elemento "
25               << i << "," << j << " da matriz:";
26          matrizA[i][j] = readInt();
27       }
28    }
29
```

```
30    //leitura dos dados das matrizes
31    cout << "Leitura da matriz B:"<<endl;
32    for( int i=0;i<dimensaoI;++i){
33       for( int j=0; j<dimensaoJ;++j){
34          cout << "Entre com o elemento "
35               << i << "," << j << " da matriz:";
36          matrizB[i][j] = readInt();
37       }
38    }
39
40    //soma das matrizes
41    for( int i=0;i<dimensaoI;++i){
42       for( int j=0; j<dimensaoJ;++j){
43          //soma das mesmas posições
44          matrizSoma[i][j] = matrizA[i][j]+matrizB[i][j];
45       }
46    }
47
48    //impressão
49    cout << "matriz A:" << endl;
50    for( int i=0;i<dimensaoI;++i){
51       for( int j=0; j<dimensaoJ;++j){
52          cout << matrizA[i][j] << "\t";
53       }
54       cout << endl;
55    }
56
57    cout << "matriz B:" << endl;
58    for( int i=0;i<dimensaoI;++i){
59       for( int j=0; j<dimensaoJ;++j){
60          cout << matrizB[i][j] << "\t";
61       }
62       cout << endl;
63    }
64
65    cout << "matriz Soma:" << endl;
66    for( int i=0;i<dimensaoI;++i){
67       for( int j=0; j<dimensaoJ;++j){
68          cout << matrizSoma[i][j] << "\t";
69       }
70       cout << endl;
71    }
72
73    cout << "Game Over" << endl;
74    return 0;
75 }
```

10.3 Subtração de matrizes

A subtração de matrizes só pode ocorrer se e somente se as matrizes forem de mesma ordem, ou seja, tenham o mesmo número de linhas e de colunas. Para fazer a subtração entre os elementos dessa matriz basta subtrair os termos correspondente de ambas, $a_{ij} - b_{ij} = c_{ij}$, que resultará em uma terceira matriz. Temos de exemplo a subtração das matrizes A e B de ordem $m \times n$ essa soma resultará na matriz C de ordem $m \times n$.

O programa 13 demonstra, em C++, como subtrair 2 matrizes:

Programa 13: Subtração de matrizes.

```
1  //subtração de matrizes
2  //programa_013.cpp
3  #include "biblaureano.h"
4
5  #define DIMENSAO_MAX 100 //dimensão máxima da matriz
6
7  int main(){
8     int matrizA[DIMENSAO_MAX][DIMENSAO_MAX];
```

```
9     int matrizB[DIMENSAO_MAX][DIMENSAO_MAX];
10    int matrizSubtracao[DIMENSAO_MAX][DIMENSAO_MAX];
11
12    int dimensaoI, dimensaoJ;
13
14    do{
15       dimensaoI = readInt("Entre com dimensão I das matrizes:");
16       dimensaoJ = readInt("Entre com dimensão J das matrizes:");
17    } while( dimensaoI <=0 || dimensaoI >= DIMENSAO_MAX ||
18             dimensaoJ <=0 || dimensaoJ >= DIMENSAO_MAX );
19
20    //leitura dos dados das matrizes
21    cout << "Leitura da matriz A:"<<endl;
22    for( int i=0;i<dimensaoI;++i){
23       for( int j=0; j<dimensaoJ;++j){
24          cout << "Entre com o elemento "
25               << i << "," << j << " da matriz:";
26          matrizA[i][j] = readInt();
27       }
28    }
29
30    //leitura dos dados das matrizes
31    cout << "Leitura da matriz B:"<<endl;
32    for( int i=0;i<dimensaoI;++i){
33       for( int j=0; j<dimensaoJ;++j){
34          cout << "Entre com o elemento "
35               << i << "," << j << " da matriz:";
36          matrizB[i][j] = readInt();
37       }
38    }
39
40    //subtração das matrizes
41    for( int i=0;i<dimensaoI;++i){
42       for( int j=0; j<dimensaoJ;++j){
43          //subtração das mesmas posições
44          matrizSubtracao[i][j] = matrizA[i][j]-matrizB[i][j];
45       }
46    }
47
48    //impressão
49    cout << "matriz A:" << endl;
50    for( int i=0;i<dimensaoI;++i){
51       for( int j=0; j<dimensaoJ;++j){
52          cout << matrizA[i][j] << "\t";
53       }
54       cout << endl;
55    }
56
57    cout << "matriz B:" << endl;
58    for( int i=0;i<dimensaoI;++i){
59       for( int j=0; j<dimensaoJ;++j){
60          cout << matrizB[i][j] << "\t";
61       }
62       cout << endl;
63    }
64
65    cout << "matriz Subtração:" << endl;
66    for( int i=0;i<dimensaoI;++i){
67       for( int j=0; j<dimensaoJ;++j){
68          cout << matrizSubtracao[i][j] << "\t";
69       }
70       cout << endl;
71    }
72
73    cout << "Game Over" << endl;
74    return 0;
75 }
```

10.4 Multiplicação de matrizes

Em matemática, o produto de duas matrizes é definido somente quando o número de colunas da primeira matriz é igual ao número de linhas da segunda matriz. Se A é uma matriz $m \times n$ e B é uma matriz $n \times p$, então seu produto é uma matriz $m \times p$ definida como AB. O produto é dado por $(AB)_{ij} = \sum_{r=1}^{n} a_{ir}b_{rj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj}$.

Considere A de dimensão 2×3:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \rightarrow A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

e B de dimensão 3×2:

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \rightarrow B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}.$$

A figura 13 ilustra a multiplicação entre as matrizes e a matriz resultante.

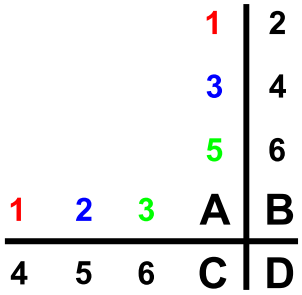


Figura 13: Multiplicação de matrizes.

Portanto, chega-se a:

- $A = a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31}$
- $B = a_{11} \times b_{12} + a_{12} \times b_{22} + a_{13} \times b_{32}$
- $C = a_{21} \times b_{11} + a_{22} \times b_{21} + a_{23} \times b_{31}$
- $D = a_{21} \times b_{12} + a_{22} \times b_{22} + a_{23} \times b_{32}$

Ou seja:

- $A = 1 \times 1 + 2 \times 3 + 3 \times 5 = 1 + 6 + 15 = 22$
- $B = 1 \times 2 + 2 \times 4 + 3 \times 6 = 2 + 8 + 18 = 28$
- $C = 4 \times 1 + 5 \times 3 + 6 \times 5 = 4 + 15 + 30 = 49$
- $D = 4 \times 2 + 5 \times 4 + 6 \times 6 = 8 + 20 + 36 = 64$

Portanto, a matriz resultante $(A \times B)$ é: $AB = \begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix}$.

O programa 14 demonstra, em C++, como multiplicar 2 matrizes:

Programa 14: Multiplicação de matrizes.

```
1  //multiplicação de matrizes
2  //programa_014.cpp
3  #include "biblaureano.h"
4
5  #define DIMENSAO_MAX 100 //dimensão máxima da matriz
6
7  int main(){
8     int matrizA[DIMENSAO_MAX][DIMENSAO_MAX];
9     int matrizB[DIMENSAO_MAX][DIMENSAO_MAX];
10    int matrizMultiplicacao[DIMENSAO_MAX][DIMENSAO_MAX];
11
12    int dimensaoIA, dimensaoJA;
13
14    do{
15       dimensaoIA = readInt("Entre com dimensão I das matriz A:");
16       dimensaoJA = readInt("Entre com dimensão J das matriz A:");
17    } while( dimensaoIA <=0 || dimensaoIA >= DIMENSAO_MAX ||
18             dimensaoJA <=0 || dimensaoJA >= DIMENSAO_MAX );
19
20    //leitura dos dados das matrizes
21    cout << "Leitura da matriz A:"<<endl;
22    for( int i=0;i<dimensaoIA;++i){
23       for( int j=0; j<dimensaoJA;++j){
24          cout << "Entre com o elemento "
25               << i << "," << j << " da matriz:";
26          matrizA[i][j] = readInt();
27       }
28    }
29
30    int dimensaoIB, dimensaoJB;
31
32    cout << "O número de colunas da matriz A devem ser igual ao "
33            "número de linhas da matriz B:" << endl;
34    dimensaoIB = dimensaoJA;
35
36    do{
37       dimensaoJB = readInt("Entre com dimensão J das matriz B:");
38    } while( dimensaoJB <=0 || dimensaoJB >= DIMENSAO_MAX );
39
40    //leitura dos dados das matrizes
41    cout << "Leitura da matriz B:"<<endl;
42    for( int i=0;i<dimensaoIB;++i){
43       for( int j=0; j<dimensaoJB;++j){
44          cout << "Entre com o elemento "
45               << i << "," << j << " da matriz:";
46          matrizB[i][j] = readInt();
47       }
48    }
49
50    //multiplicação de A por B
51    for( int i=0;i<dimensaoIA;++i){
52       for( int j=0; j<dimensaoJB;++j){
53          int soma = 0;
54          //aqui ocorre a multiplicação e soma dos elementos
55          for( int k=0; k<dimensaoJA; ++k){
56             soma += ( matrizA[i][k] * matrizB[k][j] );
57          }
58          matrizMultiplicacao[i][j] = soma;
59       }
60    }
61
62    //impressão
63    cout << "matriz A:" << endl;
64    for( int i=0;i<dimensaoIA;++i){
65       for( int j=0; j<dimensaoJA;++j){
66          cout << matrizA[i][j] << "\t";
67       }
68       cout << endl;
69    }
70
71    cout << "matriz B:" << endl;
```

```
72      for( int i=0;i<dimensaoIB;++i){
73          for( int j=0; j<dimensaoJB;++j){
74              cout << matrizB[i][j] << "\t";
75          }
76          cout << endl;
77      }
78
79      cout << "matriz Multiplicação:" << endl;
80      for( int i=0;i<dimensaoIA;++i){
81          for( int j=0; j<dimensaoJB;++j){
82              cout << matrizMultiplicacao[i][j] << "\t";
83          }
84          cout << endl;
85      }
86
87      cout << "Game Over" << endl;
88      return 0;
89  }
```

10.5 Exercícios

1. Em uma confecção são produzidos três modelos de calças: A, B e C. Sendo usado dois tipos de botões G (grande) e M (médio). O número de botões usado por modelo de calça é dado pela seguinte tabela:

	Calça A	Calça B	Calça C
Botões P	6	4	2
Botões G	4	3	2

   O número de calças produzidas nos meses de novembro e dezembro é fornecido pela tabela a seguir:

	Novembro	Dezembro
Calça A	60	100
Calça B	80	90
Calça C	70	120

   De acordo com os dados fornecidos, faça um programa que calcule a quantidade de botões gastos nos meses referidos.

2. Dadas duas matrizes numéricas A e B de dimensão 4×3, fazer um programa que gere uma matriz lógica C, tal que o elemento $C[i][j]$ seja verdadeiro se os elementos nas posições respectivas das matrizes A e B forem iguais, e falso caso contrário. Exibir as matrizes A, B e C.

   Exemplo: $A = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 5 & 9 \\ 3 & 7 & 2 \\ 4 & 6 & 8 \end{bmatrix}$ e $B = \begin{bmatrix} 2 & 5 & 8 \\ 1 & 9 & 7 \\ 3 & 7 & 1 \\ 4 & 5 & 8 \end{bmatrix}$

   então $C = \begin{bmatrix} true & false & false \\ true & false & false \\ true & true & false \\ true & false & true \end{bmatrix}$

3. Elaborar um programa que lê uma matriz M[6][6] e um valor A e multiplica a matriz M pelo valor A e coloca

os valores da matriz multiplicados por A em um vetor de V[36] e escreve no final o vetor V.

4. Escrever um programa que lê uma matriz A[15][5] e a escreva. Verifique, a seguir, quais os elementos de A que estão repetidos e quantas vezes cada um está repetido. Escrever cada elemento repetido com uma mensagem dizendo que o elemento aparece X vezes em A.
5. Escrever um programa que lê uma matriz M[10][10] e a escreve. A seguir, troque a diagonal principal com a diagonal secundária. Apresente a matriz modificada.
6. Na teoria dos sistemas, define-se como elemento *minimax* de uma matriz o menor elemento da linha onde se encontra o maior elemento da matriz. Escreva um programa que leia uma matriz 10×10 de números e encontre seu elemento *minimax*, mostrando também sua posição.
7. Escrever um programa que lê uma matriz M[5,5] e cria 2 vetores SL[5] e SC[5] que contenham, respectivamente, as somas das linhas e das colunas de M. Escrever a matriz e os vetores criados.
8. Faça um programa lê uma matriz A 7×7 de números e cria 2 vetores ML[7] e MC[7], que contenham, respectivamente, o maior elemento de cada uma das linhas e o menor elemento de cada uma das colunas. Escrever a matriz A e os vetores ML e MC.
9. Altere o programa da batalha naval para que ele preencha randomicamente o tabuleiro com os navios. Lembre-se que não pode haver navios na mesma posição.
10. Escrever um programa que monte um campo minado. O programa deve perguntar as dimensões da matriz (considere no máximo uma matriz de 20x20) e a quantidade de minas. Lembre-se que neste jogo, cada quadrado vazio contém um número que indica quantos quadrados minados estão próximos (encostados) a ele.