

JOGO DA VELHA

Entendendo um pouco mais da linguagem C

Hoje iremos nos aprofundar um pouco mais na programação em C. Iremos construir um pequeno jogo da velha sem inteligência artificial (movimentos aleatórios). Para começar vamos pensar como isso será feito.

No jogo da velha temos um campo com três linhas e três colunas e cada jogador marca sua jogada até que um deles consiga três marcações na mesma linha, coluna ou em uma das diagonais.

Já tem alguma idéia de como faremos isso? Pois vou lhe dizer: matrizes! O campo nada mais será do que uma matriz 3X3 do tipo inteiro, onde os campos com valor 0 são os campos livres, os com valor 1 serão as jogadas do jogador e as com valor 2 serão as jogadas do jogo. Ficará mais ou menos assim:

0	1	2
2	2	1
1	1	2

Mas ficaria confuso e até chato para o jogador jogar com valores numéricos, por isso criaremos uma nova matriz 3X3 do tipo caractere para mostrar o jogo como ele é na realidade, então o jogo acima se parecerá com isto:

	X	O
O	O	X
X	X	O

Bem melhor, não? Certo e como faremos para trabalhar com as duas tabelas ao mesmo tempo? A resposta é simples: funções! Neste tutorial mostrarei como criar suas próprias funções e fazer com que interajam entre si. Precisaremos de uma função que analise qual campo o jogador deseja marcar e coloque o valor 1 na posição correspondente na primeira matriz. Depois, uma nova função irá fazer com que o jogo marque um campo qualquer, desde que já não esteja marcado e por fim, uma última função irá analisar os valores da primeira matriz e marcar o símbolo correspondente nos campos da segunda.

Certo, já sabemos como o jogo irá funcionar, mas ainda não temos como saber se o jogador ganhou ou perdeu. Então vamos analisar a matriz para entender como fazer isso:

	0	1	2
0		X	O
1	O	O	X
2	X	X	O

Então... já sabe como resolver o problema? Se ainda não sabe então vou contar: precisamos de um vetor auxiliar de três posições que irá armazenar o

valor de cada linha, coluna e diagonal, um de cada vez e verificar se os três valores são iguais. Ele estará em uma função que irá encerrar o jogo e mostrar uma mensagem dizendo quem ganhou ou continuar o jogo caso os valores não sejam iguais. Agora devemos pensar quais valores armazenar neste vetor:

Linhas:	j	j	j
Colunas:	i	i	i
Diagonal principal:	$i=j$	$i=j$	$i=j$
Diagonal sec.:	$i+j = 2$	$i+j = 2$	$i+j = 2$

A lógica disso é simples, basta pensar como a linguagem trabalha com matrizes:

```
for(i=0; i<3; i++){
    for(j=0; j<3; j++){
        ...
    }
}
```

Com o código acima, estamos trabalhando em todas as colunas de cada linha. Se estivermos na primeira linha, trabalhamos com todas as colunas e aí passaremos à próxima linha para fazer o mesmo.

Digo que as linhas são j, j, j , pois são todas as colunas da linha em que estou trabalhando. As colunas são i, i, i , porém não é assim que funciona, na verdade teremos de inverter o valor de i e j na verificação e fazer o mesmo procedimento que fizemos para as linhas para conseguir, então não trabalharemos mais todas as colunas de cada linha e sim todas as linhas de cada coluna. Quando o valor de i e j são iguais temos um elemento da diagonal principal e quando sua soma é igual a 2, temos um elemento da diagonal secundária.

Então comecemos a implementar nossa idéia. Antes de mais nada devemos criar um projeto na nossa IDE e depois incluir as bibliotecas. As que iremos usar são:

```
stdio.h, conio.h, stdlib.h
```

Então nossas primeiras três linhas de código ficam assim:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

Depois das inclusões, vamos criar a função principal:

```
main() {
```

```
}
```

Antes de prosseguir devemos aprender alguns conceitos sobre variáveis. Elas podem ser tanto globais quanto locais. Uma variável local só pode ser acessada dentro da função em que está contida e é reiniciada cada vez que a função é chamada, isso inclui a função principal. As variáveis globais podem ser usadas por qualquer função e não é reiniciada durante a execução (a menos que você programe isto), porém elas ocupam mais espaço em memória quando o programa está sendo executado.

Visto isso, devemos pensar quais os dados que não queremos perder durante a execução, ou seja: as duas matrizes. Todos os demais valores podem ser facilmente recriados. Então, vamos declarar as duas matrizes como globais, para isso basta coloca-las no início do código e antes de qualquer outra função, desta forma:

```
|inclusões|  
  
int matriz_logica[3][3];  
char matriz_visual[4][4];  
  
|função principal|
```

Feito isto, você deve ter percebido que a matriz visual é maior que a lógica. Isto é uma regra de sintaxe da linguagem C, pois nela os vetores de caracteres são formados pelos caracteres (3) e por um caractere especial ('\0') que indica o fim do vetor, logo temos quatro posições. Se você criar um vetor de 3X3 na verdade terá um de 2X2, então atenção a este detalhe!

Agora outro detalhe da linguagem: apesar de nossa matriz lógica ser do tipo inteiro, ela não é inicializada com 0, e sim com valores aleatórios chamados de "lixo de memória", isto vale para qualquer variável de qualquer tipo, então nossa primeira tarefa será criar uma função que irá inicializar nossas duas matrizes com os valores que queremos. Um detalhe: cada função é de um tipo, este tipo está relacionado ao tipo de dado retornado pela função, um exemplo:

```
int somar(int a, int b){  
    int c;  
    c = a+b;  
    return c;  
}  
  
main(){  
    int p, s;  
    puts("Informe o primeiro numero: ");  
    scanf("%d", &p);  
    puts("Informe o segundo numero: ");  
    scanf("%d", &s);
```

```
printf("Resultado %d", somar(p,s));
getch();
}
```

Temos uma função do tipo inteiro que retorna como resultado o valor da variável `c`. Esta função possui dois parâmetros inteiros, porém não necessariamente uma função do tipo inteiro usa somente parâmetros inteiros, como já foi dito, o tipo da função depende do retorno que ela dá. A função soma os dois números que foram passados por parâmetro na função principal. Note que as variáveis não precisam ter o mesmo nome, por isso usei `a` e `b` na função soma e `p` e `s` na função principal. Isto acontece pois ambas as funções usam variáveis locais.

Então qual tipo de função usaremos para zerar ambas as matrizes que são de tipos diferentes? Teríamos que criar duas funções diferentes, uma de cada tipo, certo? Não deixa de estar errado, mas como nossas matrizes são variáveis globais, podemos usar uma função de tipo indefinido e que não retornará nenhum valor, somente alterará o valor das matrizes. Então nossa primeira função ficará assim:

```
|inclusões|
|matrizes|
int i,j;

void inicia_matrizes(){
    for (i=0; i<3; i++){
        for (j=0; j<3; j++){
            matriz_logica[i][j] = 0;
        }
    }

    for (i=0; i<4; i++){
        for (j=0; j<4; j++){
            matriz_visual[i][j] = ' ';
        }
    }
}

|função principal|
```

Como estamos trabalhando com matrizes, iremos também declarar `i` e `j` como globais para não precisarmos declará-los localmente em todas as nossas funções. Nossos `for(j..)` não possui um bloco (`{}`), pois contém somente uma instrução e como vimos em nossa primeira lição o bloco só é necessário quando uma função possui mais de uma instrução. Definimos o valor 0 para todos os campo da nossa matriz lógica e um valor em branco (`' '`) para toda a nossa matriz visual. Assim já temos como começar a trabalhar nas demais funções. A próxima delas é a função que irá mostrar a matriz visual na tela e perguntar ao jogador qual campo deseja marcar. Ela fica assim:

```

|inclusões|
|matrizes e contadores|
int x, y;

|inicia_matrizes|

void mostra_tela(){
    system ("cls");

    puts(" * * * JOGO DA VELHA * * *");
    for (i=0; i<4; i++){
        printf ("\n");
        for (j=0; j<4; j++){
            printf ("%c", matriz_visual[i][j]);
            if (j < 4)
                printf (" ");
        }
    }

    printf ("\n\n");
    puts("Informe a linha: ");
    scanf("%d", &x);
    puts("Informe a coluna: ");
    scanf("%d", &y);
    x -= 1;
    y -= 1;
}

```

Antes do laço de repetição temos uma função que limpa a tela via comando MS-DOS, como visto na lição anterior, também pulamos uma linha antes de mostrar os campos de cada linha e caso a coluna seja qualquer uma, exceto a última imprimimos um espaço em branco para separar melhor os campos. Mais adiante pulamos duas linhas para mostrar as mensagens e ler as coordenadas e nas duas últimas linhas temos algo estranho: `x -= 1`. Lembra-se do `i++`, que soma 1 ao valor de `x`? Esta operação é semelhante, ela subtrai o número 1 do valor de `x`. É o mesmo que: `x = x - 1` e fazemos isto, pois é de se imaginar que nossos jogadores não sabem que as colunas começam do 0 e vão até o 2, se ele quer a primeira linha, então informará 1 e não 0, por isto sempre iremos diminuir 1 dos valores informados. Note que adicionei `x` e `y` junto às variáveis globais para não precisar passá-las por parâmetro.

Certo, já sabemos qual campo o usuário deseja marcar, então está na hora de trabalhar na função que irá fazer isto:

```

void marca_campo(int quem){
    matriz_logica[x][y] = quem;
}

```

```
}
```

Esta função será usada para marcar tanto as jogadas do jogador quanto do jogo, por isto usei um parâmetro inteiro que receberá o valor 1 ou 2 conforme o caso. Veremos isto mais adiante, pois agora devemos gerar duas coordenadas aleatórias para as marcações do jogo, isto é feito assim:

```
void turno_jogo() {
    int a, b;
    gerar:
        a = rand()%3;
        b = rand()%3;

    if (matriz_logica[a][b] != 0)
        goto gerar;
    else
        x = a;
        y = b;
}
```

Esta função gera dois números aleatórios menores que 3 e verifica se a coordenada correspondente a eles já não está marcada. Se estiver, vai reiniciar a função até que um campo ainda não marcado seja selecionado. Nomeamos como `gerar`, a linha a partir da qual iremos retomar caso necessário, isso é feito pelo comando `goto`.

A esta altura você deve estar pensando: “Mas x e y não estão recebendo valores tanto do jogador quanto do jogo e não estão sendo usados para nada?” A resposta é SIM, mas só por enquanto. Por estarmos usando funções, a forma de programar é diferente. Não iremos chamar a função que permite ao jogador escolher um campo e a que marca um campo aleatório ao mesmo tempo, elas irão seguir uma ordem, então o jogo irá funcionar, tenha calma.

Certo, a próxima função é que irá atualizar a matriz visual com os símbolos correspondentes às marcações feitas, então aqui vai:

```
void atualiza_visual() {
    for (i=0; i<3; i++){
        for (j=0; j<3; j++){
            if (matriz_logica[i][j] == 1)
                matriz_visual[i][j] = 'X';
            if (matriz_logica[i][j] == 2)
                matriz_visual[i][j] = 'O';
        }
    }
}
```

Agora a última função, ela será responsável por verificar se o jogador ganhou ou perdeu. Ela fica assim:

```

void verifica_vitoria(){
    int campos[3];
    bool ganhou = false;
    bool perdeu = false;

    for (i=0; i<3; i++){
        for (j=0; j<3; j++){
            campos[j] = matriz_logica[i][j];
        }
        if ((campos[0] == campos[1]) && (campos[1] ==
campos[2])){
            if (campos[0] == 1)
                ganhou = true;
            if (campos[0] == 2)
                perdeu = true;
        }
    }

    for (i=0; i<3; i++){
        for (j=0; j<3; j++){
            campos[j] = matriz_logica[j][i];
        }
        if ((campos[0] == campos[1]) && (campos[1] ==
campos[2])){
            if (campos[0] == 1)
                ganhou = true;
            if (campos[0] == 2)
                perdeu = true;
        }
    }

    if ((matriz_logica[0][0] == matriz_logica[1][1]) &&
(matriz_logica[2][2] == matriz_logica[0][0])){
        if (campos[0] == 1)
            ganhou = true;
        if (campos[0] == 2)
            perdeu = true;
    }

    if ((matriz_logica[0][2] == matriz_logica[1][1]) &&
(matriz_logica[2][0] == matriz_logica[1][1])){
        if (campos[0] == 1)
            ganhou = true;
        if (campos[0] == 2)
            perdeu = true;
    }

    if (ganhou == true){

```

```

        system ("cls");
        puts("Voce ganhou!");
        getch();
        exit(1);
    }

    if (perdeu == true){
        system ("cls");
        puts("Voce perdeu!");
        getch();
        exit(1);
    }
}

```

Esta função faz o que comentamos no começo, ela verifica se os valores do vetor de campos são iguais e qual o valor destes campos. Se o jogador ganhar a função limpa a tela e mostra a mensagem, logo em seguida o jogo é fechado, o mesmo para o caso em que o jogo ganha.

Agora vamos colocar tudo para funcionar. Vamos até a função principal e digitar o seguinte:

```

main() {
    inicia_matrizes();

    while (true){
        mostra_tela();
        marca_campo(1);
        turno_jogo();
        marca_campo(2);
        atualiza_visual();
        verifica_vitoria();
    }
}

```

Vamos entender o que fizemos aqui. Antes de mais nada chamamos a função que inicializa as matrizes, depois definimos um laço infinito de repetição com a seqüência do jogo, onde a função `marca_campo` recebe o parâmetro correspondente a cada turno.

Finalmente nosso jogo da velha está jogável. Agora cabe a você implementar algumas coisinhas extras como por exemplo verificar se o usuário informou realmente um número e não uma letra e se o número está entre 1 e 3. Divirta-se!