PROJETO EM CONJUNTO COM AS DISCIPLINAS DE TÉCNICAS E LABORATÓRIO DE DESENVOLVIMENTO

Jogo optado: Jogo da velha 3x3

Nome dos integrantes:

Matheus Farias Porto Dos Anjos

RGM: 31259821

Ryan Do Nascimento Bezerra

RGM: 31268684

Fábio Harley Silva Filho

RGM: 30290759

Gabriel Henriques Cassiano

RGM: 32096143

Introdução:

O jogo da velha é um jogo para dois jogadores, a estrutura do jogo é apenas um tabuleiro 3x3 onde os jogadores marcam 'X' ou 'O' em cada campo do tabuleiro. Para vencer o jogo, um dos jogadores deve preencher uma linha, uma coluna ou uma diagonal inteira com 'X', ou 'O', o primeiro que fizer isso vence, e o jogo termina, caso o tabuleiro fique totalmente preenchido e ninguém ganhar, teremos um empate.

Descrição geral do jogo:

O jogo da velha desenvolvido possui um menu inicial onde o jogador pode escolher:

Jogar, no qual se localiza mais duas opções de escolha, uma para o modo jogador vs jogador, e outra para o modo jogador vs computador

Ver Ranking Geral, no qual se localiza o log de todas as partidas jogadas após a primeira execução do script

Créditos, no qual se localiza o nome dos desenvolvedores do algoritmo

Sair, no qual se localiza a opção de terminar a execução do script

Para jogar, a cada jogada precisamos fornecer dois números que representam a posição do tabuleiro em que queremos jogar: a coluna e a linha, nessa ordem. As linhas, bem como as colunas, variam de 0 até 2 e somente os locais vazios podem ser escolhidos como jogada.

Demonstração do código:

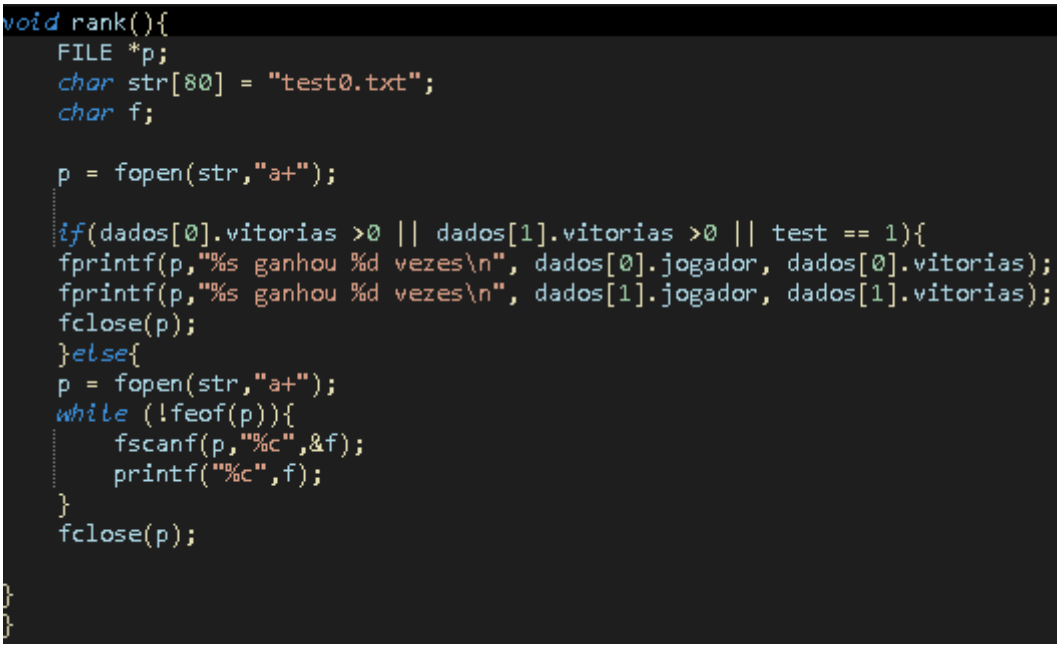
```c
do{
    printf("\n %s - Coloque a coluna: ", dados[i].jogador);
    retorno = scanf("%d",&c);
    printf(" %s - Coloque a linha: ", dados[i].jogador);
    retorno = scanf("%d",&l);
do{
    letra = getchar();
    printf("%c", letra);
    }while(letra != '\n');
}while(retorno == 0 || c < 0 || c >= 3 || l < 0 || l >= 3);



if((matriz[l][c] == 'O') || (matriz[l][c] == 'X')){
    printf("\nPosicao Ocupada");
    Sleep(1000);
    jogadas--;
    break;
}else if(jogadas %2 == 1){
    matriz[l][c] = 'X';
    sair = 1;
}else if(jogadas %2 == 0){
    matriz[l][c] = 'O';
    sair = 1;
}
```

Tivemos que limpar o buffer do teclado para evitar o loop infinito no caso do jogador digitar sem querer alguma letra e isso pode ser resolvido com a função getchar(). O último dado no buffer do teclado sempre será um “\n”, que representa a tecla ENTER. Assim, basta ler caractere por caractere enquanto o caractere lido for diferente de “\n”, como apresentado no trecho de código a seguir, limpando o buffer e dando oportunidade para o usuário digitar novamente. O while de retorno == 0 foi modificado conforme os limites impostos por nós, como visto acima, caso o jogador digite qualquer número diferente de 0, 1, 2, haverá o retorno de colocar a linha e a coluna novamente

Uma área de armazenamento temporário é chamada de **buffer**. Todos os dispositivos de entrada e saída padrão contêm um **buffer** de entrada e saída.



```
void rank(){
    FILE *p;
    char str[80] = "test0.txt";
    char f;

    p = fopen(str,"a+");

    if(dados[0].vitorias >0 || dados[1].vitorias >0 || test == 1){
    fprintf(p,"%s ganhou %d vezes\n", dados[0].jogador, dados[0].vitorias);
    fprintf(p,"%s ganhou %d vezes\n", dados[1].jogador, dados[1].vitorias);
    fclose(p);
    }else{
    p = fopen(str,"a+");
    while (!feof(p)){
        fscanf(p,"%c",&f);
        printf("%c",f);
    }
    fclose(p);

}
}
```

O “**void rank()**” foi definido como uma função que salva os dados de vitórias dos jogadores/computador na partida armazenados em uma struct, esses dados são salvos no arquivo “ranking” toda vez que uma nova partida é concluída, se encerrada pela opção “Sair (4)” (Partidas de uma execução anterior da atual também ficam salvas com o intuito de ser um ranking geral de todas as partidas já jogadas)

Um dos problemas que tivemos foi relacionado ao “BOT” que jogaria contra o usuário caso não houvesse segundo jogador. Inicialmente, pensamos em apenas fazer com que esse bot gerasse números aleatórios de 0 a 2, tanto para linha quanto para coluna, assim, definindo o local da jogada do computador. Segue a imagem:

```
while(sair == 0){
    if(bot ==1 && jogadas %2 == 0){

    srand (time(NULL)); //BOT
    l = rand() % 3 + 0;
    c = rand() % 3 + 0;

    bot_tentativa++;

    if(bot_tentativa > 5){
        for(l = 0; l < 3; l++){
            for(c = 0; c < 3; c++)
                if(matriz[l][c] == ' '){
                    matriz[l][c] = 'O';
                    l = 3;
                    c = 3;
                    bot_tentativa = 0;
                    sair = 1;
                }

        }

    }else if(matriz[l][c] == 'O' || matriz[l][c] == 'X'){
        jogadas--;
        break;
    }else if(jogadas %2 == 0){
        matriz[l][c] = 'O';
        sair = 1;
```

Para que isso fosse possível, utilizamos a função ***srand()*** para geração de números aleatórios, mas após alguns testes, notamos que, na verdade, gerar números aleatórios não é tão simples assim: o maior dos problemas era que a própria função ***srand()*** não gerava números aleatórios. Em todos os testes, os números gerados eram os mesmos, e no fim, o computador nunca jogava, pois o local da jogada já estava ocupado.

Mais tarde, descobrimos ser necessário definir uma **“semente”** para a função ***srand()***, o pontapé inicial para que a função começasse a gerar números **diferentes**. Precisávamos que essa semente mudasse a todo momento, para isso, usamos uma nova função: **time(NULL)** dentro da função **srand()**. Essa função retorna os segundos desde 1 de Janeiro de 1970 até o horário local da máquina que está rodando o código, dessa forma, a **semente** muda a cada segundo que é passado.

Mesmo depois disso, ainda existia um limitador que impedia que o código rodasse corretamente: a probabilidade que a função gerava números diferentes era bem baixa, acreditamos ser devido à pouca liberdade de escolha de números pela função, na qual definimos de 0 até 2. A solução implementada foi fazer com que, caso o computador tenha 6 ou mais tentativas de jogadas, e ainda assim, não foi possível encontrar um local vazio, forçamos o BOT a percorrer toda a matriz da tabela 3x3 utilizando 2 “**for()”** até que ele encontre o primeiro espaço vazio no jogo da velha.