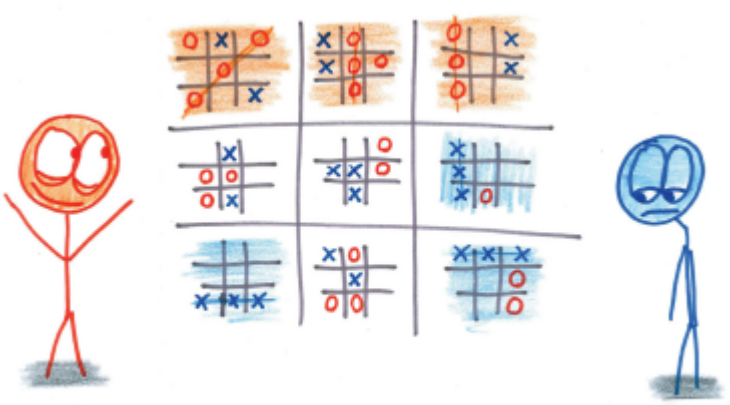
Relatório de Programação -"Ultimate Tic-Tac-Toe"



Realizado por:

Daniel Ferreira Rodrigues a2021142013

> *Indice*

> Descrição genérica da organização do programa

O programa do jogo como no enunciado do trabalho prático diz tem algumas regras, umas pedidas pelo enunciado e outras que implementei.

O jogo começa com um par ou ímpar para decidir quem começa e quem decide qual o primeiro mini tabuleiro por onde começar. O jogador assinala as coordenadas onde pretende colocar o seu carácter no mini tabuleiro, essas coordenadas que servirão de seguida para informar o programa que o próximo mini tabuleiro será nessas coordenadas do tabuleiro grande.

Basicamente o jogo consiste num jogo do galo mais complexo com 9 tabuleiros, quando se completa um dos mini tabuleiros é assinalado então no tabuleiro principal, quando um desses tabuleiros é completado, fica "inativo" passando sempre para o seguinte.

O jogo acaba quando estiverem 3 mini tabuleiros completos em diagonal, horizontal ou vertical assinalados pelo mesmo caractere, quando o jogo acaba um ficheiro de texto é criado com informações de todas as jogadas feitas.

Antes de cada jogada, o jogador pode guardar o jogo e retomar mais tarde a esse jogo, como também pode rever algumas jogadas anteriores que estejam entre 1 e 10 e superiores ao número de jogadas feitas por exemplo:(Total:4 só pode rever entre 1 a 4 jogadas).

Cada jogada feita é guardada dentro de uma lista ligada, para que o jogador possa rever jogadas anteriores.

O programa tem 8 ficheiros diferentes, sendo main.c a função principal, matdin.h aproveitei e adicionei funções "gerais", utils.h funções random que usei, retornaJogo.h funções de retornar o jogo, guardajogo.h funções que permitem guardar informação e guardar o jogo e criar o ficheiro de texto, 1vcpu.h funções usadas para o jogo contra o

computador, 2jogadores.h funções usadas para o jogo de 2 jogadores e jogadasant.h contém as funções para poder criar e gerir uma lista ligada.

> *Identificação do ambiente de desenvolvimento utilizado durante a implementação*

Para a implementação deste trabalho utilizei o ide Code::Blocks, que para mim é um ide simples e contém tudo o que preciso enquanto iniciante nesta área da programação.

> *Apresentação das estruturas dinâmicas implementadas e da organização dos ficheiros utilizados pelo programa, justificando as escolhas feitas*

o *jgant lista*

Ponteiro de lista ligada que inicialmente está vazio e que de acordo com cada jogada, vai armazenando informações das jogadas, após cada jogada lista.prox aponta para a próxima jogada.

Foi usado porque é um dos requisitos que se encontra no enunciado, mas também porque é uma maneira mais simples de poder gerir o que foi feito em jogadas anteriores, mantendo essa lista ligada ordenada pelo número de jogadas.

Com esta lista ligada podemos rever o que foi feito em jogadas anteriores, saber por exemplo o que o jogador x fez em y, em [c1][c2].

No final de cada jogo esta lista é libertada.

> *Justificação para as opções tomadas em termos de implementação, nomeadamente na definição de algumas das regras do jogo.*

A primeira decisão foi criar então uma estrutura de dados para guardar o tabuleiro de jogo e os 9 mini tabuleiros.

De seguida criei um menu com 4 opções onde o utilizador pode escolher:1-Individual, 2-2 Jogadores,3-Carregar Jogo e 4-Sair.

Aproveitando o ficheiro matdin.c e matdin.h que o professor forneceu adicionei novas funções "gerais" que pudessem ser úteis para todo o código, os tabuleiros criei com as funções fornecidas por esse ficheiro.

Em relação a algumas regras de jogo criei algumas funções para que me simplifica-se algum código e implementei novas estratégias.

Para o modo de jogo contra o computador utilizei as funções que geram números aleatórios dentro de um limite fornecidos pelo professor no ficheiro utils.h.

Quando se começa um jogo é necessário passar pelo jogo do par ou ímpar onde depois os utilizadores 1 valor cada 1, é inserido uma variável auxiliar que soma esses dois valores e depois vai dividindo esses valores até essa variável ser igual a 1 ou a 0, 1 é ímpar e 0 é par, o vencedor tem direito a escolher o primeiro mini tabuleiro.

O caso de saber qual seria o próximo mini tabuleiro optei por criar uma função que lesse as coordenadas da jogada anterior feita pelo jogador, essas coordenadas que depois são colocadas na função proximominitab(int x, int y), que devolve o número do próximo mini tabuleiro que se encontra no array mini tabuleiro minitabs[9].

A cada jogada chamo a função validaespaco(char **p, int x, int y) para saber se o jogador pode ou não introduzir nas coordenadas que deseja e após ser validado esse

espaço, chamo a função verifica(char **p, int x, int y) para fazer uma verificação no mini tabuleiro atual e no tabuleiro principal.

Sempre que alguém ganha um mini tabuleiro é colocado o caractere desse usuário no tabuleiro principal e tabuleirogrande.minitabs[x].seccaovencida toma o valor de 1 para identificar que esse mini tabuleiro está então "inativo", e as próximas coordenadas baterem com esse mini tabuleiro passar para o mini tabuleiro ao lado.

O jogo acaba após a chamada da função verifica(char **p, int x, int y) devolver ou então o número de jogadas ser igual a 81(caso de empate).

O ficheiro jogadasant.h guarda uma struct e todas as funções necessárias para criar a minha lista ligada simples para que o jogador possa rever jogadas anteriores.

Inicialmente há um ponteiro que aponta para o primeiro elemento da lista onde é inserido os dados necessários da primeira jogada e de seguida o ponteiro jgant lista, aponta para a próxima lista, segundo o enunciado como o jogador só pode rever 10 jogadas antes, em mostraJogadasAnteriores(jgant t, int totaljogadasfeitas, int retornax), uso uma variável auxiliar que é o resultado da diferença entre totaljogadasfeitas e retornax(valor de jogadas que o jogador deseja rever), essa aux irá servir como um id onde irá percorrer as listas todas até que encontre a lista com o número de jogada igual a esse valor aux, de seguida é só percorrer um ciclo com o número de vezes que o utilizador deseja.

Para guardar o jogo criei uma struct com alguns dados necessários para saber que funções usar como o id do próximo jogador ou então o modo de jogo que o jogo foi salvo. Para além dessas informações tem um array saveInfoJogagada minitabsinfo[81] que guarda a informação de todas as jogadas feitas.

Essa struct é escrita depois para o ficheiro "jogo.bin", quando o jogo é retornado existe uma função do tipo save que lê os dados existentes de "jogo.bin" e armazena nessa struct, de seguida o jogo é construído desde o inicio ou seja desde

saveInfoJogagada minitabsinfo[0] até save.totalJogadas lê o modo de jogo que o jogo foi feito e chama a função ao respectivo modo.

Assim que alguém ganhar para além da função que elimina as listas ser chamada, também chamada uma função que escreve num ficheiro de texto algumas informações e todas as jogadas feitas ao longo do jogo. A estratégia foi mais ou menos como o que eu disse acima, primeiro perguntar ao utilizador o nome que deseja para o ficheiro, que depois é concatenado com ".txt", abrimos o ficheiro em modo de escrita e escrevo algumas informações de seguida chamo a função que escreve as jogadas todas feitas desde o início. Primeiramente criei uma função para escrever então os tabuleiro para o formato de ficheiro de texto que está em matdin.h (escreveJogoTabs(char **p, FILE *f)). Esta função cria um tabuleiro novo e de acordo com as informações recebidas de save.minitabsInfo[x] vai escrevendo jogada a jogada nesse tabuleiro.

Para identificar os jogadores 0 é o jogador 1 e 1 é o jogador 2 ou o pc, assim como o modo Solo é representado por 0 e o modo 2 Jogadores por 1.

Para saber qual é o próximo jogador é basicamente se recebo 0 então o próximo é 1 e se receber 1 o próximo é 0.