Trabalho Prático de Programação

Jogo do Semáforo

Nome: Bruno Sousa a2020132971@isec.pt

DEIS-LEI

Docente: Francisco Pereira

Índice

1. Introdução

Este trabalho de programação tem como base a criação do jogo do semáforo, cujo é feito usando a linguagem de programação C, na norma C99.

Para fazer este trabalho foi necessário o uso estruturas dinâmicas, manipulação de ficheiros,etc.

2. Código Fonte

O código fonte do programa encontra-se dividido por cinco ficheiros de código (extensão .c) e quatro ficheiros do tipo *header* (extensão .h):

- main.c

  Neste ficheiro, é executado todo o código que roda o programa, ou seja, é aqui que são executadas todas as funções que constroem o jogo.
- utils.c/.h

  Ficheiro de suporte fornecido pelos professores da disciplina. Tem funções que permitem escolher aleatoriamente um número num intervalo de números, podendo ser usado este mecanismo na criação do BOT.
- Textos.c/.h

  Este ficheiro apenas contem alguns menus e funções que iteragem com os jogadores para obter DATA para o programa.
- Files.c/.h

Neste ficheiro encontram-se funções que permitem a manipulação de ficheiros, sejam binários ou de texto.

- Tabuleiro.c/.h

  Este ficheiro contem todas as funções que permitem o desenvolvimento do jogo e do tabuleiro, assim como a sua manutenção. Todas as funções referentes às diferentes jogadas se encontram aqui e até mesmo funções que criam listas ligadas e desalocam memória.

3. Estruturas de Dados

Neste trabalho apenas foi utilizada uma estrutura de dados "BoardS" que guarda os dados do tabuleiro ao longo do jogo através de uma lista ligada.

```
typedef struct BoardS {
  char **tabuleiro;
  char jogador;
  int IDjogada;
  int colunas;
  int linhas;
  char jogadaL;
  struct BoardS* next;
```

Esta estrutura é incializada no ficheiro "Tabuleiro.h".

4. Estruturas Dinâmicas

Ao longo deste trabalho foi implementado diversas estruturas dinâmicas, ficando no final apenas duas.

Estrutura Dinâmica – Tabela alocada dinamicamente

Esta Estrutura Dinâmica mantem o tabuleiro ao longo do jogo. Basicamente, é uma lista que aponta para uma lista de ponteiros (basicamente são as linhas do tabuleiro) que por sua vez apontam para a informação das colunas do tabuleiro. Como esta estrutura é mantida dinamicamente é possível adicionar linhas e colunas.

```
char** Cria_matriz_dinamica(char **board,int linhas,int colunas)
{

    int i,j;
    board = (char**)malloc(sizeof(char*)*linhas);
    if (board==NULL) {
        printf("Erro na alocacao de memoria");
        return NULL;

    }
    for (i =0; i<linhas;i++) {
        board[i] = (char*)malloc(sizeof(char)* colunas);
        if (board[i]==NULL){
            printf("Erro na alocacao de memoria");
            return NULL;
        }
        for (j=0;j<colunas;j++) {
            board[i][j] =' ';
        }
    }
    return board;
}
```

Esta função é responsável pela criação da tabela dinâmica alocando o espaço necessário para a lista de ponteiros e para a lista de caracteres (acaba por ser as colunas do tabuleiro).

O motivo pelo qual escolhi esta estrutura foi porque constava no enunciado, apesar de haver a opção de fazer através de uma variante de lista ligada, considerei que fosse me nos complexos fazer desta maneira, alem disso, este tema já tinha sido abordado durante uma aula Prática de Programação podendo me basear nas informações que tinha obtido dessa aula, acabou por me facilitar a implementação dessa estrutura dinâmica.

Estrutura dinâmica do tipo Lista Ligada

Esta estrutura dinâmica usa uma estrutura de dados chamada "BoardS" que está definida no ficheiro "tabuleiro.h". Ela é responsável por guardar todas as jogadas feitas ao longo do jogo, sendo assim possível ao jogador, rever as jogadas que quiser. Basicamente ao fim de cada estrutura de dados, temos o elemento struct BoardS* next que a sua função é apontar para a próxima estrutura de dados construindo

assim a lista ligada.

```c
void new_node_tabuleiro(BoardS** root, char **board,int colunas,int linhas,char jogada,char jogador,int IDjogada) {
    //obter um tabuleiro para cada no para evitar que todos os nós para o mesmo tabuleiro no final.
    char **Board_Copia;
    BoardS* new_board=  malloc(sizeof(BoardS));
    if(new_board == NULL) {
        printf("Erro na alocacao de memoria");
        exit(1);
    }

    Board_Copia = Cria_Nova_matriz_dinamica(board,linhas,colunas);


    new_board->tabuleiro = Board_Copia;
    new_board->jogadaL = jogada;
    new_board->colunas = colunas;
    new_board->linhas = linhas;
    new_board->jogador = jogador;
    new_board->IDjogada = IDjogada;
    new_board->next = NULL;

    if(*root == NULL) {
        *root = new_board;
        return;
    }

    BoardS* curr= *root;
    while(curr->next !=NULL) {
        curr = curr->next;
    }
    curr->next = new_board;
```

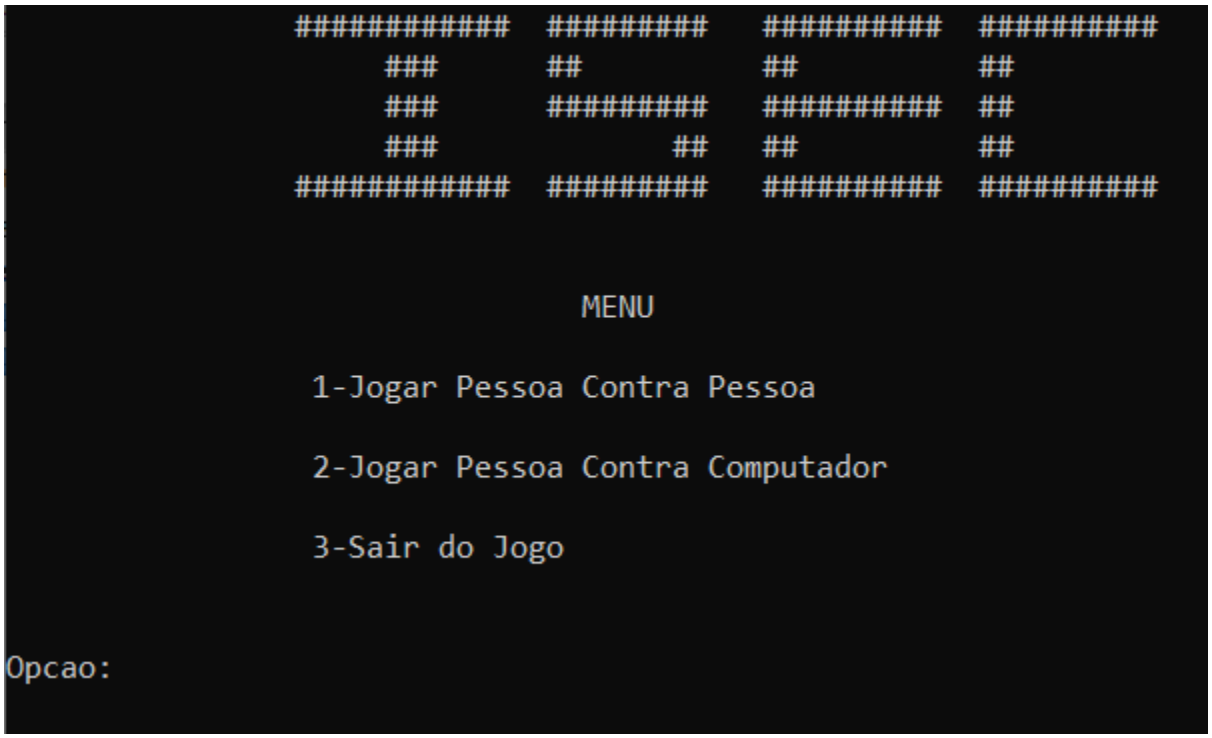
A imagem acima mostra a função responsável pela criação de um novo nó da lista ligada, sendo necessário criar um tabuleiro para cada nó, de outra maneira, o new_board->tabuleiro estaria a apontar para o mesmo tabuleiro em todos os nós, para isso é alocado espaço criando uma nova matriz dinâmica(o tabuleiro é mantido numa tabela dinâmica), apesar de não haver a necessidade de fazer este tabuleiro dinâmico pois ele será usado apenas vez e teremos de imediato os valores dos tamanhos e do próprio array, achei melhor fazê-lo dinâmico. O motivo do uso desta lista ligada foi porque é requisitado no enunciado.

Com isto acaba as estruturas dinâmicas implementadas neste trabalho. Eu iria usar um array dinâmico de estruturas para tentar fazer o ficheiro binário que salva os dados, pois seria melhor para recarregar estes dados para o jogo a partir desse array, mas acabei por ter problemas na implementação de tal, usando a lista ligada para fazer o ficheiro binário, apesar de não ter tido sucesso na recolha dos dados desse ficheiro.

4.1 O Jogo

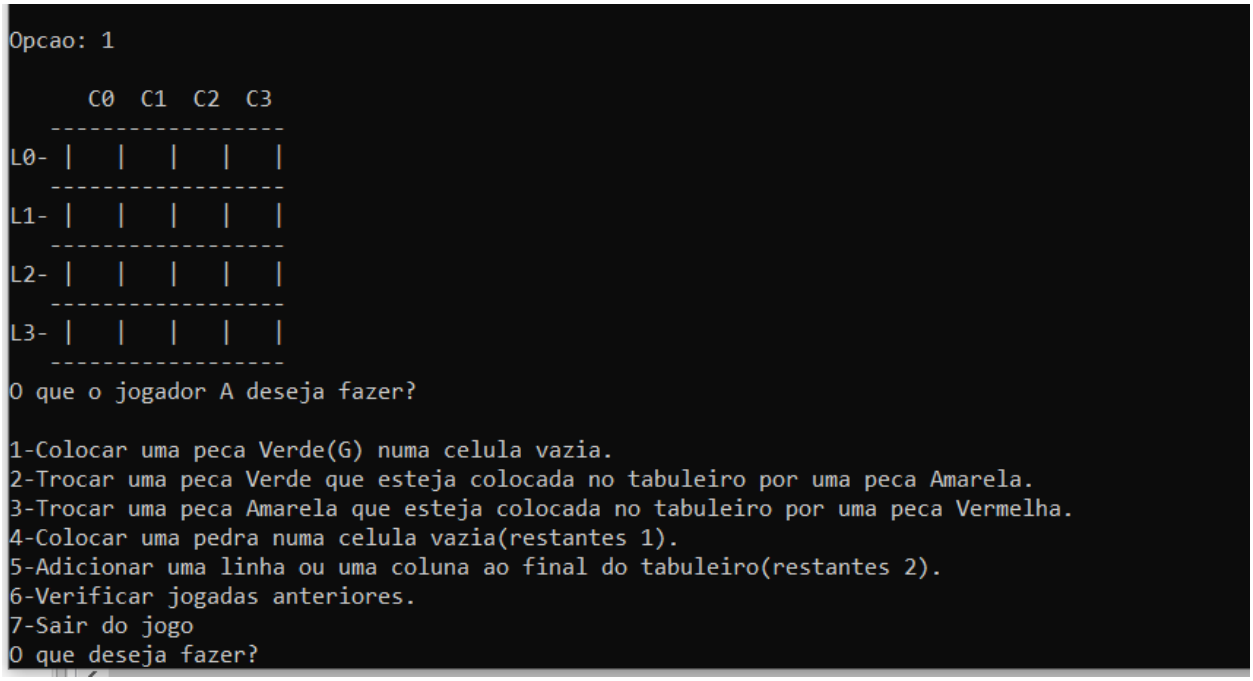
O jogo foi codificado da seguinte maneira:

- Ao começar, o utilizador é questionado através de um menu se deseja jogar contra outro jogador(opção 1) ou contra a máquina(opção 2).



No jogo Pessoa contra Pessoa, é denominado aos jogadores (“jogador A” e jogador ”B”). Assim que começa o jogo, por defeito, o jogador A começa sempre a

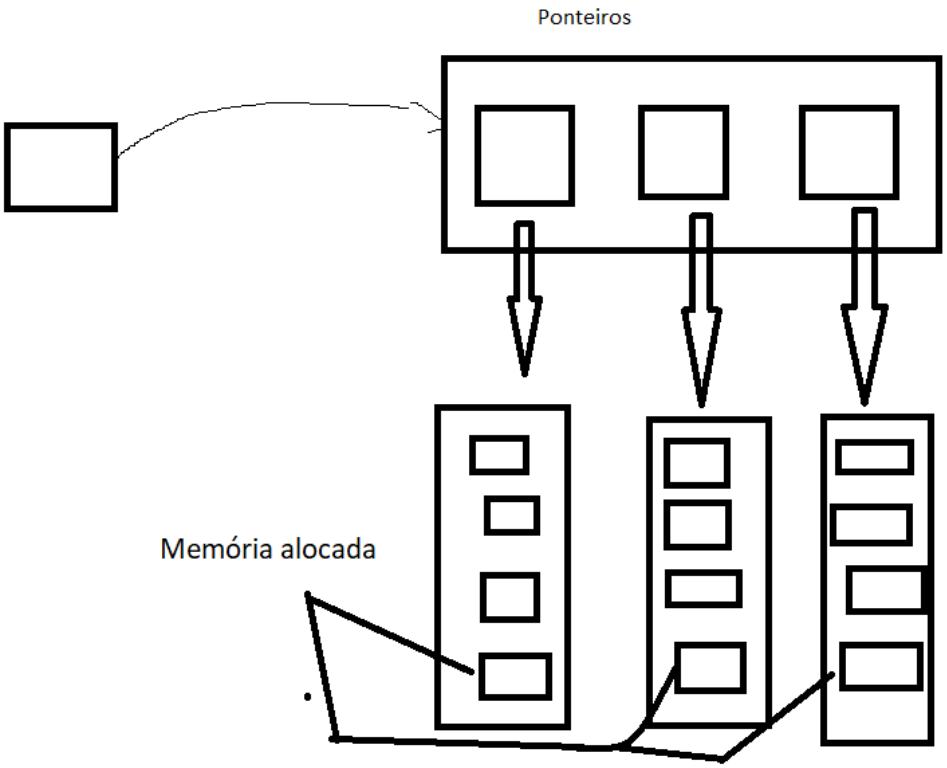
jogar e é questionado no que deseja fazer:



As funções usadas para a implementação de uma nova peça requesitam o tabuleiro usado e alteram nele o necessário para fazer a jogada pedida, se o jogador pedir para trocar uma peça verde por uma amarela, então o programa irá correr uma verificação no tabuleiro para verificar se existe alguma peça verde presente, se não, simplesmente retorna ERRO e permite ao mesmo jogador de mudar a jogada original.

Para implementar uma nova linha, tive que realocar espaço adicionando assim mais um ponteiro no array de ponteiros, depois disso aloquei memória para preencher a linha com as colunas.

Para implementar uma nova coluna, realoquei memória em cada coluna como mostra a seguinte figura:



Para mostrar as jogadas ao utilizador, fez-se uso da lista ligada, fazendo um loop pela estrutura dinâmica(a lista ligada) e imprimindo os estados do tabuleiro a partir da jogada indicada pelo jogador.

O BOT é feito apenas de ações aleatórias, sendo verificado se ação aleatória é possível ou não.

No fim o utilizador é questionado pelo nome do ficheiro que quer guardar o histórico do jogo. Sendo criado um ficheiro de texto com o nome dado pelo utilizador. A função que é responsável por esta manipulação encontra-se no ficheiro(file.c) e é a :

```
void write data(char *FILENAME, BoardS** root) ;
```

O algoritmo usado para desenhar os tabuleiros dentro do ficheiro do texto é o mesmo que é responsável por desenhar o tabuleiro no jogo.

Com isto acaba-se a explicação do código do jogo, se faltar algum elemento irei explicar na defesa se assim for necessário.

Dos itens que foram pedidos no enunciado, o único que não foi feito foi o recarregamento de um jogo depois de este ter sido interrompido.

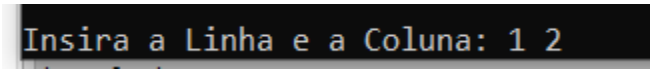
Segue-se o manual de utilização para o utilizador.

5. Manual de Utilização

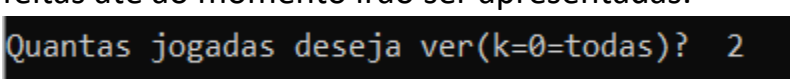
5.1 Como jogar

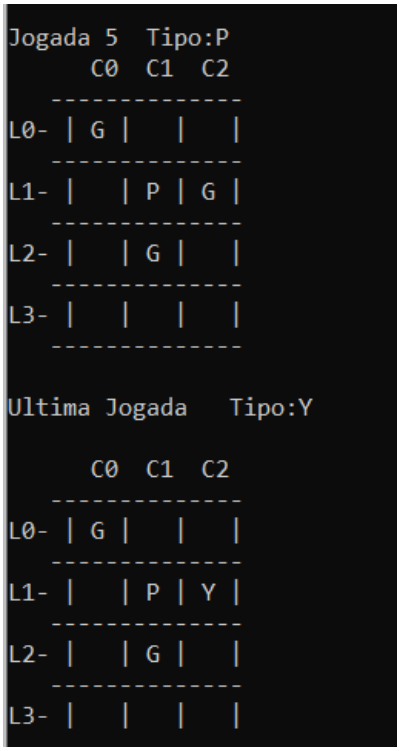
Como referido antes, é mostrado ao utilizador um menu cujo terá de escolher uma opção (1- Jogar contra pessoa, 2-Jogar contra máquina ou 3-Sair).

Depois disto o tabuleiro é inicializado, sendo o jogador da jogada do momento questionado pela jogada que deseja realizar.
O jogador deve selecionar o item adequado ao que deseja fazer, se selecionar uma jogada que necessita da linha e coluna, o jogador deve separar a informação através de um espaço por exemplo:



Para o jogador, poder ver as jogadas feitas anteriormente, tem de mencionar quantas jogadas deseja ver por ordem descendente(da última para a primeira), se o jogador inserir 0, então todas as jogadas feitas até ao momento irão ser apresentadas.

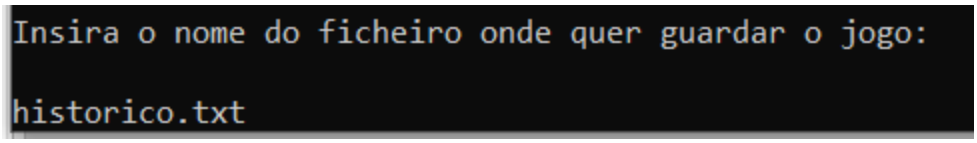




Sendo impossível o empate pelas regras do jogo, o vencedor será parabenizado pela sua vitória.

5.2 Guardar o histórico do jogo

No final do jogo o utilizador é questionado pelo nome do ficheiro para ser guardado o histórico do jogo:



Depois disto um ficheiro com este nome será criado contendo todas as jogadas realizadas durante o jogo.

Se o jogo for interrompido um ficheiro “save.bin” é criado ou atualizado com a informação necessária para o jogo recomeçar. Mas apesar da existência deste ficheiro a reconstrução das jogadas num novo jogo não foi conseguida.

Com isto acabamos o Manual de Utilização.

6. Referências

1. Moodle ISEC
2. Short introduction to linked lists in C - YouTube
3. Stack Overflow - Where Developers Learn, Share, & Build Careers
4. Binary Files - The Basics of C Programming | HowStuffWorks

Consultados em maio e junho de 2021.

7. Conclusão

Em suma, foi um projeto divertido com os seus desafios. Gostaria de ter feito um Código mais limpo e mais otimizado, mas fui tendo várias dificuldades ao longo do trabalho, acabando por não ter tempo de otimizar o projeto. Apesar disso, acho que foi um trabalho bem conseguido.

Bruno Amado Sousa a2020132971@isec.pt