
# Tecnologias e Arquitetura de Computadores
## Trabalho Laboratorial 2020/2021

Docente:
Francisco Pereira

Aluno:
João Filipe Silva de Almeida nº 2020144466 – LEI

Coimbra, 13 de Junho de 2021

# Índice

# Introdução:

O trabalho foi feito com o objetivo principal de aprimorar os meus conhecimentos da linguagem de C e me formar quanto às suas particularidades, gravação e leitura de ficheiros, manipulação de memória e manipulação de listas ligadas etc.
Tem também como objetivo secundário criar um jogo funcional que obedeça às regras do clássico jogo do semáforo.

# Estruturas de dados:

```c
typedef struct holy{
    int l;
    int c;
    int playtype;
    struct holy *prox;
}data, *pdata;

typedef struct max{
    int addlcA,addlcB;
    int RockA,RockB;
}maxplays, *pmaxplays;

typedef struct tabarr{
    char ** table;
    int ncol,nlin;
}Table, *pTable;
```

Foram usadas 3 estruturas dinâmicas para melhor organização.

A struct holy, à qual me vou referir por data ou por pdata no caso de um ponteiro para este tipo de struct, é a que vai dar origem à lista ligada que irá guardar as jogadas feitas de turno para turno.

A struct max, ou maxplays, é usada para possibilitar uma mais fácil gestão das jogadas apenas válidas uma quantidade limitada de vezes para ambos os jogadores simultaneamente.

A struct tabarr ou Table, armazena a informação relativa ao array2D dinâmico, ou seja um ponteiro para este e a quantidade de linhas e colunas que tem presentemente.

Decidi dividir desta forma pelas funções naturalmente apresentarem necessidades distintas para as variáveis relacionadas com a lista ligada e o array dinâmico bem como para a gestão das certas jogadas limite dos jogadores.

# Lista ligada:

A lista ligada usa a seguinte struct, como já explicado:

```
typedef struct holy{
    int l;
    int c;
    int playtype;
    struct holy *prox;
}data, *pdata;
```

O ponteiro inicial da lista ligada é pela primeira vez criado no princípio do programa dentro da função inicializer:

```
pdata list = NULL
```

Elementos são adicionados à lista apenas numa situação, dentro da função playoptions, onde ou a jogada do jogador ou do computador consoante a situação é guardada dentro da nova struct adicionada:
Criação do novo elemento a adicionar:

```
new = malloc(sizeof(data));
if(new==NULL){
    printf("error in malloc");
    end_game(ptable,list);
}
```

Preenchimento do novo elemento, quer pelo input do jogador e funções de sua interpretação e verificação ou pelo output da função robot que representa o jogador automático:

```
do{
    (*new).playtype = 0;
    if (!(turn%2==0 && robot1==1)){
        fgets(saveinput,7,stdin);
    } else
        robot(saveinput,ptable,pmaxplays);
    (*new).playtype = interpretplay(saveinput, &(*new).l, &(*new).c, pmaxplays, turn, ptable,robot1);
    if((*new).playtype==0 && isrobotplayin(turn,robot1)==0){
        printf("\terror in input, try again\n");
    }
} while ((*new).playtype==0 );
(*new).prox=NULL;
```

O novo elemento é adicionado ao início da struct para mais fácil manipulação posterior:

```
if(list==NULL){
    list=new;
} else {
    aux=list;
    while(aux->prox != NULL)
        aux=aux->prox;
    aux->prox=new;
}
```

**Os outros locais que iram utilizar as listas ligadas serão:**

A função totext, só chamada no final do jogo, que irá gravar a sucessão de jogadas para um ficheiro de texto:

```
void totext(pdata p,int robot1){
    int count=1;
    char fichtxt[50]="a";
    FILE *fp;
    printf("Choose file name to save the succession of all plays: ");
    fgets(fichtxt,sizeof(fichtxt),stdin);
    fp = fopen(fichtxt,"wt");
    if (fp == NULL) {
        printf("failed to save plays to file\n");
    } else {
        fprintf(fp,"-------------------------\nPlaytype examples:\nplaytype 1: 1,2\nplaytype 2: 1,2S\nnplaytype 3: C\nnplaytype 4: L\n\nplayer B is a bot:
        %s\n-------------------------\n\n",robot1==1 ? "TRUE" : "FALSE");
        while (p!=NULL){
            fprintf(fp,"round %d, player %c: \nl=%d, c=%d, playtype=%d\n",count++,whosplaying(count),(*p).l,(*p).c,(*p).playtype);
            p = p -> prox; // left pointer beggins pointing to right pointer
        }
    }
    fclose(fp);
}
```

A função pausar, que irá gravar a lista ligada para um ficheiro binário fich.bin:

```
void pause(pTable ptable,pdata list,int robot1,int initdim,pmaxplays pmaxplays){
    FILE * fp;
    int i;
    printf("writing game state to fich.bin...");
    fp = fopen("fich.bin","wb");
    if (fp == NULL) {
        fclose(fp);
        end_game(ptable,list);
    };
    if (list==NULL){
        i=0;
        fwrite(&i,sizeof(int),1,fp); //To indicate that there is no data to read
        printf("no game data to save, aborting\n");
    } else {
        i=1;
        fwrite(&i,sizeof(int),1,fp); //To indicate that there is data to read
        fwrite(&robot1,sizeof(robot1),1,fp);
        fwrite(&initdim,sizeof(initdim),1,fp);
        fwrite(pmaxplays,sizeof(maxplays),1,fp);
        //show(list);
        while (list!=NULL && list->playtype!=0){
            fwrite(list,sizeof(data),1,fp);
            list = list -> prox;
        }
    }
    //Binary save:
    //[Int flag wether there's valid data, 0 or 1][robot1][initdim][pmaxplays)][linked list]
    fclose(fp);
    end_game(ptable,list);
}
```

A função initializer para decriptar o ficheiro binário:

```
tmp=0;
do{
    pnew = malloc(sizeof(data));
    if(pnew==NULL){ end_game(ptable,list); }
    pnew->playtype=0;
    if(tmp==0)
        list = pnew;
    else
        listaux->prox=pnew;
    listaux2 = listaux;
    listaux = pnew;
    tmp=1;
} while ( (fread(pnew, sizeof(data), 1, fp)) == 1);
free(listaux);
listaux2->prox = NULL;  //eliminate last garbadge struct
```

Sem ser nestas situações, a lista ligada é passada a várias outras funções com a finalidade de ser libertada  através do free, caso haja uma falha numa outra manipulação de memória ou de ficheiros.

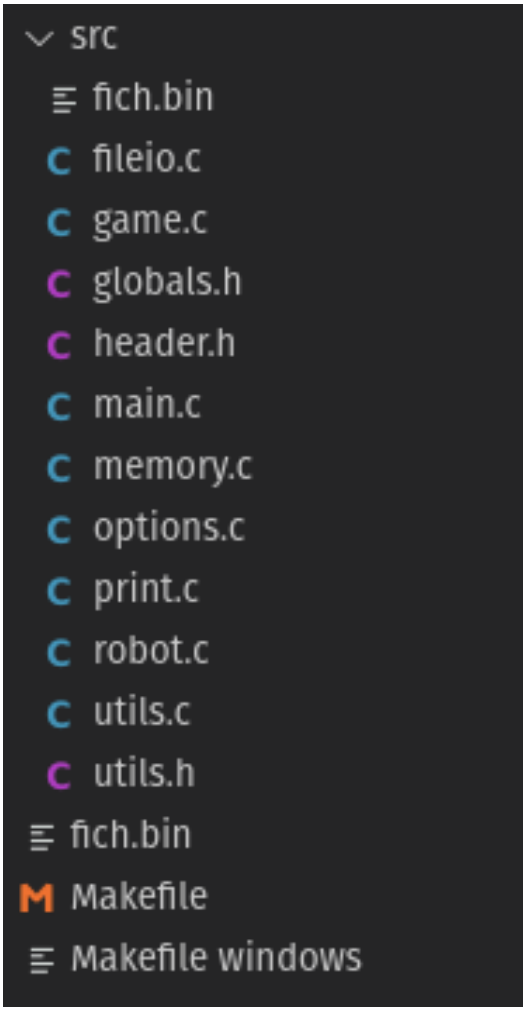
# Planeamento e Implementação:

No ficheiro main.c, é possível encontrar um planeamento inicial de como iria abordar o projeto:

```
/*
Plano Inicial (first draft):
Função Inicializar o Tabuleiro com 3-5 linhas/colunas aleatóriamente (quadrado) (usar initRandom();)
   -Verificar se temos um ficheiro binário "jogo.bin", perguntar se quer retomar o jogo.
   -Função Perguntar se quer jogar contra outro jogador ou contra um automático
Função Escrevê-lo na consola
Função Turno - diz se é jogador A ou B
   - chamada da função jogada
       -Perguntar "qual é a sua jogada?"" na consola
       -Receber input
           -Escolha: Jogar(1), Ver tabuleiros anteriores(2), Pausar(3)
               Pausar(3):
                   -guardar informação relevante num ficheiro binário "jogo.bin"
               Ver tabuleiros anteriores(2)
                   -se o utilizador indicar K=3, o programa
apresenta na consola a sucessão de estados do tabuleiro nas 3 últimas jogadas.
               Jogar(1):
                   -do tipo 1,2
                   -1,2P no caso de Pedra
                   -ou L (linha nova no fim do tabuleiro)
                   -ou C (coluna nova no fim do tabuleiro)
                   -  b (escolher outra opção, voltar atráz)
       - chamada da função restrições que diz se é válida ou não
           -se não válida, obriga a jogar outra vez
   - chama a função determina_resultado_jogo
           -condições que verificam se alguma diagonal / linha / coluna está preenchida com a mesma cor
- vitória - guarda num ficheiro de texto com nome dado pelo utilizador e apaga "jogo.bin"
            -Nota de rodapé: Se o tabuleiro não for quadrado, será impossível terminar o jogo formando
uma diagonal toda da mesma cor.
Função restrições
   -válido:
       -jogador que joga pode:
           -jogar verde numa casa vazia
           -jogar amarelo numa casa verde
           -jogar vermelho numa casa amarela
           -jogar uma pedra numa casa vazia
           -Criar uma coluna ou linha
               -no caso de criar uma coluna ou linha, criar no máximo 2 destes por jogo
               -no caso de pedra, máximo 1 vez por jogo
Função determina_resultado_jogo

Tabuleiro 4*4:
S - Stone
---------------
| G : Y :   : R |
|---:---:---:---|
| S : G :   : Y |
|---:---:---:---|
|   :   :   :   |
|---:---:---:---|
| S : G : Y : R |
---------------
*/
```

O projeto foi dividido numa multitude de ficheiros, com a finalidade de organizar e organizar e tornar mais clara a sua leitura:



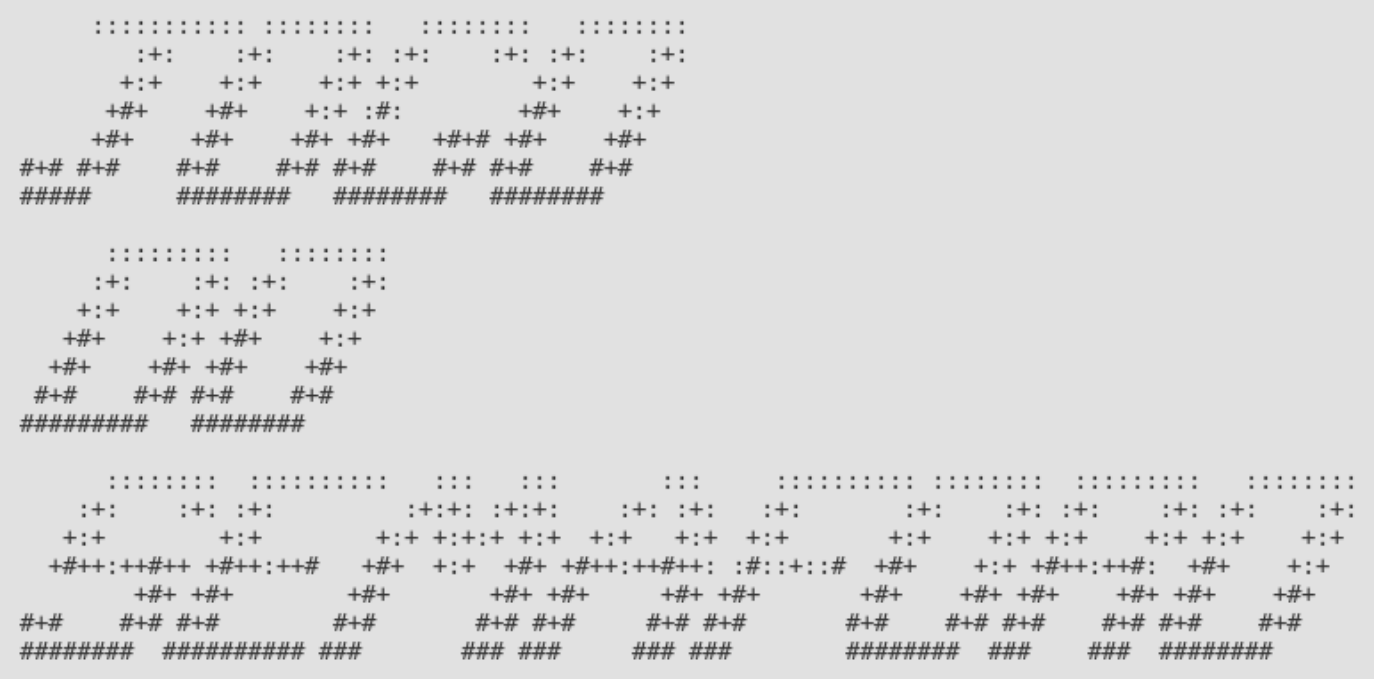

Foi ainda criado um ficheiro Makefile para facilitar a compilação do projeto na sua integridade.

**fileio.c** contém as funções relativas à manipulação de ficheiros.
**game.c** contém as funções relativas à gestão do jogo.
**main.c** contém a main.
**memory.c** contém as funções relativas à gestão de memória.
**options.c** contém as funções relativas ao input do jogador e sua interpretação.
**print.c** contém as funções com a finalidade de imprimir para o ecrã dados.
**robot.c** contém as funções relacionadas com o jogador automático
**header.h** é um header inicializar as funções usadas
**globals.h** é um header para as structs usadas.
**utils.c** foi dada
**utils.h** foi dada

Em geral no projeto as funções construídas foram abordadas com o objetivo de ser possível a sua reutilização em quando oportuno em várias partes do código de forma a torná-lo mais sucinto.

# Manual de Utilização:

O jogo começa com um ecrã para  apresentar o jogo:



As regras do jogo são explicadas neste video: Semáforo

São apresentados no terminal explicações sucintas de como proceder em cada passo do programa.

Se um ficheiro binário for encontrado, o utilizador deverá decidir carregá-lo ou Não escrevendo S ou N e entre no terminal, para Sim e Não respetivamente.

Deverá escolher jogar contra outro jogador ou contra um robô com 2 ou 1 respetivamente.

De seguida 1,2,3 ou 4 poderão ser dados como input para Jogar, ver jogadas anteriores, pausar, ou ver um menu de ajuda respetivamente.

O menu de ajuda explica como é que as jogadas deverão ser executadas:

```
       How to play:
Valid inputs:

line,column - Places the apropriate piece in the square with the given coordinates if possible
Example: 1,2    places piece in the first line second column

line,columnS - Places a stone in the square with the given coordinates if possible (max one per player)
Example: 1,2R   places rock in the first line second column

C - Adds a column,
L - Adds a line
         each player can play one of these two options twice per game

Objective: compleate a line, column or diagonal of the same color (diagonal are only possible on square tables)
```