Licenciatura em Engenharia Informática

2020/2021

Relatório

Trabalho Prático de Programação

Trabalho realizado por:
Gustavo Mateus - 2020138902

Distribuição do Código:

main.c -> Contém apenas a função main, servindo de lógica principal do programa, através do qual todas as outras funções são chamadas.

input_files.c -> Contém as funções relativas a inputs e à manipulação de ficheiros.
input_files.h

tabuleiro.c -> Contém as funções relativas à manipulação e print do tabuleiro.
tabuleiro.h

player_hist.c -> Contém as funções relativas ao player, bot, vencedor e à história.
player_hist.h

utils.c -> Contém funções relativas à geração de números aleatórios.
utils.h

structs.h -> Responsável por manter as structs histórico e dados_jogo, fazendo as mesmas disponíveis para todos os outros header files.

Apresentação da principal estrutura de dados:

```
struct dados_jogo{
    int ylenght;
    int xlenght;
    int nPedrasB;
    int nPedrasA;
    int nAumentosB;
    int nAumentosA;
    int current_player;
};

struct dados_jogo jg;
```

Esta estrutura permite acesso às variáveis cruciais para a gestão do jogo de uma maneira organizada, sendo constantemente atualizada para representar o estado atual do mesmo.

Interage com:

changeTabuleiro
ChangeCurrentPlayer
checkForWinner
ResizeTabuleiro
InicializaTabuleiro
Input1
Input2
AdicionaAoHistorico
randomPlayer

Apresentação detalhada das estruturas dinâmicas implementadas:

Int **tabuleiro:

Segura o tabuleiro bidimensional do jogo.
Cada célula possui um valor INT, sendo 0-Vazia, 1-Verde, 2-Amarelo, 3-Vermelho, 4-Pedra.

Este tabuleiro é alocado dinamicamente conforme as necessidades, sendo capaz de se expandir para as dimensões desejadas através do uso da função `ResizeTabuleiro`.

O tamanho inicial do tabuleiro é determinado aqui como um valor aleatório de 3 a 5 linhas e colunas:

```
initRandom();
int tamanhoInicial = intUniformRnd(3, 5);
```

No início da execução do programa é alocada memória conforme o tamanho inicial da seguinte maneira:

```
tabuleiro = (int**)malloc(sizeof(int*)*jg.ylenght);
if (tabuleiro!=NULL){
    for (int i = 0; i<jg.ylenght; i++){
        tabuleiro[i] = (int*)malloc(sizeof(int)*jg.xlenght);

        if(tabuleiro[i]==NULL){
            printf("Erro na alocacao de memoria\n");
        }
    }
}
else{
    printf("Erro na alocacao de memoria\n");
}
```

A função `ResizeTabuleiro` recebe como argumento ‘C’ ou ‘L’ para aumentar colunas ou linhas respetivamente.

Para aumentar uma coluna:

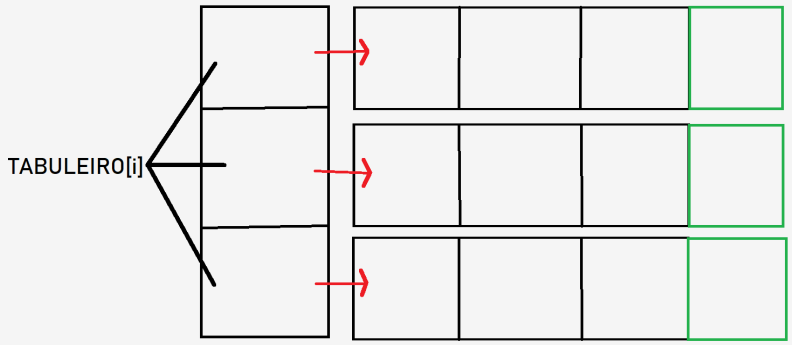
```
for(int i=0;i<(*ylenght);i++){
    tabuleiro[i] = realloc( tabuleiro[i], sizeof(int) * ((*xlenght)+1) );

    if(tabuleiro[0]==NULL){
        printf("ERRO NA ALOCACAO DE MEMORIA X%d\n",i);
        return NULL;
    }
}
```

Sendo as células a verde as que pretendemos aumentar para formar a coluna.

Esta linha refere-se individualmente à casa da lista mais à esquerda, que aponta para o início de cada linha e através do realloc aumenta o seu tamanho para suportarem mais uma célula no seu final.

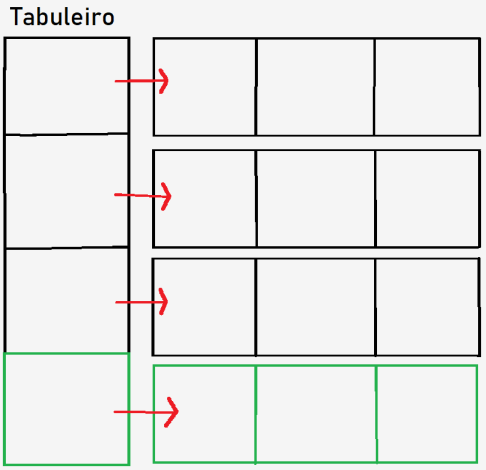
```
tabuleiro[i] = realloc( tabuleiro[i], sizeof(int) * ((*xlenght)+1) );
```



Da mesma maneira, para aumentar uma linha:

```
tabuleiro = realloc( tabuleiro, sizeof(int*) * ((*ylenght)+1) );
if(tabuleiro==NULL){
        printf("ERRO NA ALOCACAO DE MEMORIA Y1\n");
        return NULL;
}
tabuleiro[*ylenght] = malloc( sizeof(int) * ((*xlenght)) );
if(tabuleiro[(*ylenght)]==NULL){
    printf("ERRO NA ALOCACAO DE MEMORIA Y2\n");
    return NULL;
}
```

Aumentamos o tamanho do tabuleiro de forma a acomodar mais uma célula no seu fim e acedendo a essa célula alocamos memória suficiente para acomodar uma célula por cada coluna existente.



Struct historico:

```
struct historico{
    int xlenght;
    int ylenght;
    char current_player;
    int **tabuleiro;

    struct historico *next;
};

struct historico *head = NULL;
```

O histórico é uma estrutura usada para gerar uma lista ligada.
Cada node desta lista guarda o estado do jogo no momento em que a função `AdicionaAoHistorico` é chamada.

É possível visualizar esta lista invocando a função `PrintHistorico`.

Interage com:
`PrintHistorico`
`Input3`
`AdicionaAoHistorico`

`AdicionaAoHistorico` funciona criando uma instância do struct histórico (*new_histórico)

```
struct historico *new_historico = (struct historico*) malloc(sizeof(struct historico));
```

Dentro dessa instância aloca memória para o tabuleiro da mesma maneira que foi feito previamente na função main ao `**tabuleiro` e copia os conteúdos do `**tabuleiro` para o tabuleiro do new_historico.

```
for(int i=0; i<jg.ylenght; i++){
    for(int j=0; j<jg.xlenght; j++){
        new_historico->tabuleiro[i][j] = tabela_atual[i][j];
    }
}
```

Copia também as dimensões e jogador atual da estrutura dados_jogo para o new_historico.

```
new_historico->xlenght=jg.xlenght;
new_historico->ylenght=jg.ylenght;
new_historico->current_player=jg.current_player;
```

De seguida cria uma instância “last” e iguala-a à “head”.
Se a “head” for nula passa a igualar o new_historico, caso contrário faz do “last” o último node na lista ligada e aponta o penúltimo para o node que criamos agora.

Justificação para as opções tomadas em termos de implementação:

Uma opção tomada para manter o programa livre de instruções que o tornem específico para um determinado ambiente/plataforma de desenvolvimento foi eliminar o uso de fflush(stdin), substituindo-o com o **`clear_buffer`**.

```
void clear_buffer(){
    int i;
    do{
        i = getchar();
    }while ( i != '\n' && i != EOF );
    return;
}
```

```
//fflush(stdin);
clear_buffer();
scanf("%20s",NomeFicheiro);
```

Devido a constrangimentos de tempo não me foi possível implementar a funcionalidade de salvar o estado do jogo para um ficheiro .bin e recuperar-o mesmo, voltando ao estado anterior do jogo.
Mesmo assim disponibilizo como código comentado as funções `read_bin` e `write_bin` que desenvolvi parcialmente no ficheiro input_files.c.

Resumo da funcionalidade das funções mais complexas:

changeTabuleiro -> Dependendo do valor do argumento `int pedra` que lhe for passada coloca uma peça da cor apropriada ou uma pedra nas coordenadas indicadas, mas verificando antes se a jogada é legal.

checkForWinner -> Analisa o tabuleiro, procurando nas linhas, colunas e diagonais por uma sequência não interrompida de peças com cores iguais que seja do tamanho necessário para preencher a linha/coluna/diagonal na sua totalidade, independentemente das dimensões do tabuleiro.

InicializaTabuleiro -> Percorre o tabuleiro colocando a zero todas as células que não tenham já 0,1,2,3 ou 4 como valor dentro, permitindo assim descartar o lixo inerente com a alocação de memória.

Input1 -> Recebe o input do jogador, apenas aceitando como resposta 'A'/'a' , 'U' / 'u' , 'P'/'p' e 'J'/'j'.
Verifica também no caso de pedras e aumentos que o jogador atual ainda pode efetuar essas jogadas, impedindo-as de acontecer caso contrário.

Input2 -> Recebe o input do jogador, apenas aceitando como resposta posições de coordenadas (x y) que sejam válidas para o estado atual do tabuleiro.
Caso sejam válidas passa-as para a função **changeTabuleiro**.

Input3 -> Mostra o número de jogadas já decorridas e recebe o input do jogador sobre quantas jogadas este pretende visualizar, apenas aceitando como resposta um número que seja igual ou superior a zero e menor ou igual ao número de jogadas decorridas.
Quando obtiver esse número passa-o para a função **PrintHistorico** em conjunto com a head da lista ligada responsável por guardar o histórico de jogadas.

Input4 -> Recebe o input do jogador, apenas aceitando como resposta 'L'/'l' , 'C' / 'c' e devolve-o.

exportFile -> Tomando partido da lista ligada responsável por guardar o histórico de jogadas esta função abre um ficheiro de texto e escreve para dentro do mesmo a lista de todas as jogadas que decorreram, por ordem cronológica.
Desta maneira salvando um relatório completo do decorrer do jogo.

randomPlayer -> Gerando números aleatórios de 0 a 100 é possível associar uma determinada probabilidade a cada uma das ações possíveis. Dessa maneira e verificando que a jogada

selecionada é legal no tabuleiro atual esta função gera valores e chama as funções necessárias para efetuar jogadas aleatórias sem input do jogador.

Pequeno manual de utilização:

O jogo do semáforo desenrola-se num tabuleiro inicialmente vazio e dividido em células. Um jogador de cada vez coloca uma peça de Verde (G), Amarela (Y) ou Vermelha (R). Ganha o jogador que coloque uma peça que permita formar uma linha, coluna ou diagonal completa com peças da mesma cor.
Jogadas Válidas:

1. Colocar uma peça Verde numa célula vazia
2. Trocar uma peça Verde que esteja colocada no tabuleiro por uma peça Amarela
3. Trocar uma peça Amarela que esteja colocada no tabuleiro por uma peça Vermelha
4. Colocar uma pedra numa célula vazia. Máximo de 1 vez por jogo.
   A colocação de uma pedra inviabiliza que o jogo possa terminar por preenchimento da linha e coluna afetadas (e, eventualmente também da diagonal ou diagonais).
5. Adicionar uma linha ou uma coluna ao final do tabuleiro. Esta jogada adiciona linhas ou colunas completas e vazias ao tabuleiro de jogo. Máximo de 2 vezes por jogo.

Determine se quer jogar contra o computador ou contra outro jogador.

```
Escolha entre 1 ou 2 jogadores '1' ou '2':
```

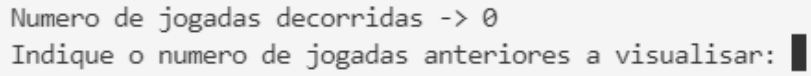
Considerando o estado do tabuleiro e o número de pedras e aumentos ainda disponíveis indique o que pretende fazer na sua ronda.

```
      x0   x1   x2   x3
     --------------------
y0  |    |    |    |    |
     --------------------
y1  |    |    |    |    |
     --------------------
y2  |    |    |    |    |
     --------------------
y3  |    |    |    |    |
     --------------------

Vez do jogador A
[ 1 Pedra e 2 Aumento(s) disponiveis ]

indique o que pretende:
        Ver as ultimas jogadas (U)
        Colocar uma pedra (P),
        Aumentar o tabuleiro (A)
        Jogar(J)
        Sair(S)
        -> 
```

Caso pretenda ver as últimas jogadas, introduza ‘U’ / ‘u’ e de seguida o número de jogadas que pretende visualizar, não excedendo o número de jogadas decorridas.



Caso pretenda colocar uma pedra ou uma côr, introduza ‘P’/‘p’ e ‘J’/‘j’ respetivamente e de seguida as coordenadas onde a mudança deve ser feita, separando a coordenada x da y por um espaço, como indicado no exemplo.

Por fim, caso pretenda aumentar o tabuleiro, introduza ‘A’/‘a’ e de seguida escreva ‘L’/‘l’ para aumentar uma linha e ‘C’/‘c’ para aumentar uma coluna.